PROGRAM PRODUCTION USING PYTHON.

-Computer Programmer Today

PROF. FRANK APPIAH

OTHER BOOKS

TITLE: Cybernetics: Cat-Mice System http://*lulu.com*

Ebook, Paperback and Hardcover (2019).

TITLE: Simulation and Modeling: Cat-Mice System

http://kdp.amazon.*com* , *http://lulu.com*

Ebook, Paperback and Hardcover (2019).

TITLE: Cybernetics Intelligence:Natural System

http://kdp.amazon.*com* , *http://lulu.com* , *http://blurb.com*

Ebook, Paperback and Hardcover (2019).

TITLE: Cybercat Simulation in Python

http://kdp.amazon.*com* , *http://lulu.com*

Ebook, Paperback and Hardcover (2019).

FRANK APPIAH, Professor.

Author:

Prof. Frank Appiah hold studentships at the Bsc. (Bachelor in computer engineering, *Kwame Nkrumah University of Science and Technology*) in 2008, Msc.(software engineering, *King's College London*) in 2009/10 and 2 doctorates of philosophy (DPhil / PhD) in computer engineering from *Kwame Nkrumah University of Science and Technology in 2013.*

Preface

This is an introduction to program production based on program design methodology. Programs are written in python language. These are AI test programs written in python language along with other languages like SQL for cyber-intelligence concepts.

Program production describes production of programs divided into two phases namely the problem solving phase – *problem defintions*, *algorithm design* and implementation phase – *actual program coding*.

Testing and debugging takes place during these phases which will be looked at. This book summarizes the design and maintenance of computer progams developed in the AI tests.

Table of Contents

Preface.....7
CHAPTER 1 13
INTRODUCTION 13
TO 13
CYBERINTEL.....13
PRODUCTION.....13
1. Interpreter Program (Read-Eval-Print Technique).....14
1.1 Problem Solving Phase.....17
1.1.1 Problem Definitions.....17
1.1.2 Algorithm Design.....23
1.1.2.1 Algorithmic Notations.....24
1.1.2.2 Algorithm 50
1.2 Installing.....67
1.3 Summary.....69
CHAPTER 2 71
PYTHON.....71
PROGRAMMING 71
OF 71
CYBERINTEL.....71
2.0 Cyberintel Implementation Phase.....72
2.1 Program Codes.....81
2.2 Testing and Debugging.....114
2.2.1 Program Testing.....115
2.2.2 Program Debugging.....125
2.3 Summary.....125
BIBLIOGRAPHY.....127
APPENDIX A.....128
.....130
APPENDIX B.....140
Idle Pseudocodes: 140
checktransition Python Shell.....140
commandline Python Shell.....145

querydatabase Python Shell.....146
printfetchdata Python Shell.....146
execTime Python Shell.....147
showCEModel Python Shell.....147
showHierarchy Python Shell.....148
showCECase Python Shell.....150
outputMenu Python Shell.....151
interactiveproof Python Shell.....152
stimulate Python shell.....154
logicprogram Python Shell.....156
writeouput Python Shell.....158
writedata Python Shell.....158
Cyberintelldev- Cyberintel AI Programalike (Othercodes).....158

Illustration Index

Illustration 1: Idle : Python Shell Programming.....73
Illustration 2: Syntax for def heading.....80
Illustration 3: readProgram: Functional Abstraction Module.....117
Illustration 4: Import Python Shell.....159
Illustration 5: Variables Python Shell.....159
Illustration 6: Main Program Python Shell.....160
Illustration 7: Delay Python Code.....160
Illustration 8: connectdatabase Python Shell.....160
Illustration 9: readProgram Definition.....161
Illustration 10: readProgram Continue 1.....161
Illustration 11: readProgram Continue 2.....162
Illustration 12: readProgram Continue 3.....162
Illustration 13: readProgram Continue 4.....163
Illustration 14: readProgram Continue 5.....163
Illustration 15: readProgram Continue 6.....164
Illustration 16: readProgram Continue 7.....164
Illustration 17: readProgram Continue 8.....165
Illustration 18: readProgram Continue 9.....165

Index of Tables

Table 1: Taxonomy of Forms of Testing [Source: James Peters & Witold Pedcrycz, Cf. 3].....117
Table 2: Cyberintel Forms of Testing.....119

CHAPTER 1

INTRODUCTION TO CYBERINTEL PRODUCTION.

What the computer can do will do?
I have already said that it will do
Do what I say for it will do?

--Appiah Academy Blog

INTERPRETATION
OF
CYBERINTEL.

1. Interpreter Program (Read-Eval-Print Technique)

This is cybernetics intelligence[4] interpreter based on cybernetics intelligence . This interpreter is based on read-eval-print technique that has 17 main programs/procedure defintions:

1. Check Transition Functions.
2. Check Transition Points.
3. Print Transitions.
4. Show Stored Views.
5. Show Cause-Effect Model.
6. Show Natural Hierarchy.
7. Show Cause-Effect Case.
8. Print Book Chapters & Summary.
9. Show Main Menu.
10. Validate and Verify Interactions.
11. Cyber-Simulation.
12. Transition Semantics.

13. Logic Program.
14. Intelligence Builds.
15. Show Textfile Session.
16. Run TextFile Simulation.
17. Press Esc and Enter to quit.

The interpreter program is stimulated at a delay of *protime* variable. It is a read-eval-print while-loop in the *readprogram* procedure definition. This is the main pseudocode of the *main* procedure task but others include readDataCSV, outputMenu, connectDatabase. The *readDataCSV* will read cat.csv file and build a copy of transitions possible in the cause-effect model. The *outputMenu* will print the 17 programs menuitems available to select from. The *connectDatabase* will establish a connection to MySQL[5] database called *cybintell*. Finally, the readprogram procedure is called. The main procedure script code is below:

```
def main():
1.    print '\n'
2.    time.sleep(protime-5);
```

```
3.    p='====================================='
4.    print p
5.    odata.append(p);
6.    p='WELCOME TO CYBER-INTELLIGENCE INTERPRETER.'
7.    print p
8.    odata.append(p);
9.    odata.append('TEXTFILE DETAILS ON
10    INTERPRETATION.');
11.   time.sleep(protime-3);
12.   print 'WRITTEN BY Dr. Frank Appiah'
13.   time.sleep(protime-2);
14.   print 'YEAR: 2019  MONTH: MAY, 1'
15.   time.sleep(protime-1);
16.   p='Version: 1.0.0'
17.   print p
18.   odata.append(p);
19.   p='====================================='
20.   print p
21.   odata.append(p)
22.   print os.getcwd()
23.   cmd=''
24.   data=readDataCSV()
25.   outputMenu();
26.   connectDatabase();
27.   readProgram(cmd, data);
28.   db.close();
```

The production of cyberintel program or any program can be divided into two phases : *problem solving phase*- problem defintion and algorithm design; and *implementation phase*- actual program coding.

1.1 Problem Solving Phase

Here, i will start with the problem defintions of the cyberintel program. There are about 17 of these definitions to be look at in this book. These are as described above in the previous section (1).

1.1.1 Problem Definitions

Let us now look at the enumerated list of cyberintel problem defintions. In such guidelines, below are the precise formulation of statements of the problem to be solved by an

algorithm:

1. **Check Transition Functions** : This program helps one to study the transition functions of a cause-effect(ce) model. Once you know the 9 transition states in ce model of a cyberintel, you can verify a state function of a transition.

2. **Check Transition Points** : This program helps one to study the transition points of a cause-effect model. Once you know about 19 or more transition path in ce model of a cyberintel, you can verify a path of model.

3. **Print Transitions**: This program prints transitions to study the transition points of a cause-effect model by transversing through the inputted path. Once you know about 19 or more transition paths in ce model of a cyberintel, you can verify a path.

4. **Show Stored Views**: There are about 12 views to help one to understand and study

the concepts of cyber-intelligence and more.

5. **Show Cause-Effect Model**: This will print the entire cause-effect model. There are about 9 transitions in the model with three branches namely cat-mice path, cat-human path and cat-physicale path.

6. **Show Natural Hierarchy**: There are about three branches as said above in (5). The program will allow you to select each and then will print the path shape of each cause-effect hierarchy.

7. **Show Cause-Effect Case**: The three ce model will be printed each as a ce case in non-hierarchical manner. A horizontal view of cause-effect branch is a cause-effect case.

8. **Book Chapters & Summary**: The chapters of the cyber-intelligence book is printed by as a chapter-by-chapter summary in an interactive manner.

9. **Show Main Menu**: This program will print the main menu of the interpreter. This helps one to select an item from the menu to interactively execute the several programs provided.

10. **Validate and Verify Interactions**: This program helps the reader or student to validate or verify cause – effect models. This is about proof of cause-effect transitions by query language. This uses what i call & ends | middles statement.
    Mathematically: & || &.
    Symbolically: and or or and.

11. **Cyber-Simulation**: This is a stimulation program that teaches prolog-alike syntax of stimulation concern here. There are about 10 stimulae in this program only. Each stimulation is provided after a protime delay or sleep.

12. **Transition Semantics**: This program prints out the semantics of a provided transition

function or paths. With the provided, a cause and an effect is concatenated into a new meaning of words. The sense of it is upto the tester to make. It is quite essential to look at the new concepts evolving from it.

13. **Logic Program**: This is the main intelligence program of the interpreter. It is a logic program that stimulates the concept of slot-frame syntax, substitution and unifications. It starts with the particular predicate print, then after a protime-3 sleep, a substituting print then after a protime-2 sleep, a slot-frames print then after a protime-2x sleep and unifications print. It then goes to read-eval-print loop.

14. **Intelligence Builds**: The intelligent builds program is made of 6 functions including cybersmart rep (read-eval-print) condition, cyberlearning rep condition, cyberextreme rep condition, cyberquick rep condition, cyberability rep condition and cybersharp rep condition. On each rep condition, a

particular intelligent build is selected to be intrepreted by first a select query on the cyberintell database and then printing the fetch data in an interactive manner.

15. **Show Textfile Session**: This an output data just as the screen ouputs during the program testing or execution. There are actually two textfiles mainly cyberdatafile and cyberoutfile. The cyberoutfile is saved on each session of running. It is named with a default-timestamp fashion.

16. **Run TextFile Simulation**: This will use the cyberdatafile to run a textfile simulation to generate an output just like the cyberoutfile saved during a session of running.

17. **Press Esc and Enter to quit**: This quits the whole interpreter and halts a program execution. In this program, the textfiles both cyberoutfile and cyberdatafile files are persisted on quiting the program. Finally, a quit procedure is called to stop the system

execution in shell session.

1.1.2 Algorithm Design

The creative part of program production is the algorithm design. The process will provide a guideline in solving a particular problem. A program is a made up of an algorithm. This is precisely a procedure task but an algorithmic format can be look at. A software is on the other hand is made up of a collection of programs. Therefore, a software is about a collection of precise procedure specifications that is correct according to a program.

Program Production:

It follows as specificed:

problem =>defintion=> algorithmic =>algorithm=>program=>software.

The production of program may not necessary follow the specification. It can be approached in

reverse or top-down design methodology.

Program Production:

It follows a top-down design:

software(library) <=>program<=> algorithm =>algorithmic=>problem=>definition.

I will follow the top-down design approach.

1.1.2.1 Algorithmic Notations

This will use the algorithmic notation of normal mathematical layouts in computations or computing- discrete algorithmic mathematics. The algorithmic notation will follow as described:

Procedure: Given a problem such that ...
Design a program to help or provide....
Define a program to provide....

ALGORITHMIC NOTATION (Notation Name)
description;
Input
variable1 [Integer>=0]

```
    case        [String:={positive, negative}]
    variable 2                  [Integer>=0]

    Algorithm  AlgoName
    if(variable 1 < 2 && variable 2=5)
then
        if('positive'=case) then
     s ← Solve-down(+sol);  //+solver
        else
     s ← Solve-up(-sol);   //-solution
designer
        endif
    endif.
    Output s
```

Procedure (Check Transition Functions):

Define a procedure to provide one to study the transition functions of a cause-effect(ce) model. Once you know the 9 transition states in ce model of a cyberintel, you can verify a state function of a transition.

ALGORITHMIC NOTATION (checkTransitionProc)

Given a cause and effect in a ce model, check if the transition of a cause to an effect is transversable;

Input

cause [String:=cat, potential, ...]

effect	[String:=potential,chase, ...]
cmd	[Integer:=1]
cmdStr	[String:=cause=>effect case]
odata	[List:={textfile outputs}]
sdata	[List:={session data inputs}]
valuepair	[String:={cause-effect pair}]
index	[Integer:=Range(1 to 9)]
dictdata	[List:={cause-effect cases}]

```
Algorithm  checkTransitionAlgo

if(cmd == 1) then
    sdata.append(cmd);
    cmdStr  ← raw_input('Which
              transition?');
    odata.append('Which transition?');
    (cause, effect)  ←
              cmdStr.split("=>");
    sdata.append(' cause='+cause+'
    effect='+effect+'  \n');
    for index in range(9)
     valuepair  ← dictdata[index+1]
     (vcause, veffect)   ←
        (str(valuepair[0]), valuepair[1])
       if(effct==veffect & cause==vcause)
        then
           s  ← '__Transition Point:
              transit('+str(cause)+',
                    '+str(effect)+')';
       else
              s  ← '__Final:No
        transition match for
        '+str(cause)+'=>'+str(effect
        ))
      endif.
  endif.
 Output printLine(s).
```

Procedure (Check Transition Points) : *Define a procedure to helps one to study the transition points of a cause-effect model. Once you know about 19 or more transition path in ce model of a cyberintel, you can verify a path of model.*

ALGORITHMIC NOTATION (checkTransitionProc)

Given a cause and effect in a ce model, allow one to check or verify a transition point;
Input
cmd [Integer>=0]

```
Algorithm  checkTransitionPoAlgo

if(cmd == 2) then
    commandLine(cmd);
endif.
Output print transition points.
```

Procedure (Print Transitions): *This defines a procedure to print transitions to study the transition points of a cause-effect model by transversing through the inputted path. Once you*

know about 19 or more transition paths in ce model of a cyberintel, you can verify a path.

ALGORITHMIC NOTATION
(printTransitionsProc)
Given a cause and effect in a ce model, print specific cases in between the transition paths;

Input
cmd [Integer>=0]
pri [Integer==1]

```
Algorithm  printTransitionAlgo
if(cmd == 3) then
   commandLine(pri=1);
endif.
Output print execTime.
Output print transition paths.
```

Procedure (Show Stored Views): *Define a procedure to the 12 views stored in a database to help one to understand and study the concepts of cyber-intelligence and more.*

ALGORITHMIC NOTATION (showStoredViewsProc)

Given a list of stored views , select one to view the data in the stored database view provided by MySQL;

Input

cmd	[Integer>=0]
views	[List:={humceview, ...}]
odata	[List]
sdata	[List]
selview	[Integer>=0]
stat	[String:='select * from views']

```
Algorithm  showStoredViewAlgo

if(cmd == 4) then
   print views;
   odata.append(views);
   sdata.append(cmd);
   selview ← raw_input(__Which view
         number?);
   stat ← 'select * from
               '+str(views[selview-1])+';'
   queryDatabase(stat)
   printfetchData();
endif.
Output printFetchData();
Output print execTime.
```

Procedure (Show Cause-Effect Model): *Define a procedure to print the entire cause-effect model. There are about 9 transitions in the model with*

three branches namely cat-mice path, cat-human path and cat-physicale path.

ALGORITHMIC NOTATION (showCEModelProc)

Show the entire ce model in a prolog-alike syntax.;

Input

cmd	[Integer>=0]
stl1	[Float]
et2	[Float]
sdata	[List]
odata	[List]

Algorithm showCEModelAlgo

```
if(cmd == 5) then
        st1  ←  time.clock();
        sdata.append(str(cmd));
        odata.append(str(cmd));
        showCEModel();
        et2  ←  time.clock();
        print newline;
        exectime();
endif.
Output print syntax;
Output print execTime.
```

Procedure (Show Natural Hierarchy): *There*

are about three branches as said above in (5). Define a procedure that will allow you to select each and then will print the path shape of each cause-effect hierarchy.

ALGORITHMIC NOTATION
(showNaturalHierProc)

Show the entire natural hierarchy shape of a specific branch of ce model;

Input

cmd	[Integer>=0]
stl1	[Float]
et2	[Float]
sdata	[List]
odata	[List]

Algorithm showNaturalHierAlgo

```
if(cmd == 6) then
    st1 ← time.clock();
    cmd ← raw_input('__Which
    natural hierarchy?');
    sdata.append(str(cmd));
    odata.append(str(cmd));
    showHierarchy(int(cmd));
    et2 ← time.clock();
    print newline;
    exectime();
```

```
endif.
Output print hierarchy_shape.
Output print execTime.
```

Procedure (Show Cause-Effect Case): *The three ce model will be printed each as a ce case in non-hierarchical manner. A horizontal view of cause-effect branch is a cause-effect case. Define a procedure for the defintions.*

ALGORITHMIC NOTATION
(showCauseEffectCaseProc)

Given a selected case, show the entire cause-effect case syntax (prolog alike) of a specific branch of ce model;

Input

stl1	[Float]
cmd	[Integer>=0]
et2	[Float]
sdata	[List]
odata	[List]

Algorithm showCauseEffectCaseAlgo

```
if(cmd == 7) then
    st1 ← time.clock();
```

```
            cmd  ← raw_input('___Which
            behavioral relationship?');
            sdata.append(str(cmd));
            odata.append(str(cmd));
            showCECase(int(cmd));
            et2  ←  time.clock();
            print newline;
            exectime();
        endif.
Output print case_syntax.
Output print execTime.
```

Procedure (Book Chapters & Summary): The chapters of the cyber-intelligence book is printed by as a chapter-by-chapter summary in an interactive manner. Define a procedure to do such task.

ALGORITHMIC NOTATION (showChaptersProc)

Show the chapters and summary of book interactively.

Input

stl1	[Float]
cmd	[Integer>=0]
et2	[Float]
sdata	[List]
odata	[List]
chapter1	[String:='chapter 1 details']
chapter2	[String:='chapter 2 details']
chapter3	[String:='chapter 3 details']
chapter4	[String:='chapter 4 details']

Algorithm showChaptersAlgo

```
if(cmd == 8) then
    st1 ← time.clock();
    cmd ← raw_input('___Which
    behavioral relationship?');
    sdata.append(str(cmd));
    odata.append(str(cmd));
    cmd ← raw_input('\n___Press Enter for
    Chapter 1\n>>');
    print chapter1;
    cmd ← raw_input('\n___Press Enter for
    Chapter 2\n>>');
    print chapter2;
    cmd ← raw_input('\n___Press Enter for
    Chapter 3\n>>');
    print chapter3;
    cmd ← raw_input('\n___Press Enter for
    Chapter 4\n>>');
    print chapter4;
    et2 ← time.clock();
    print newline;
    exectime();
```

```
endif.
Output print chapters from 1 to 4.
Output print execTime.
```

Procedure (Show Main Menu): *Define a procedure that will print the main menu of the interpreter. This helps one to select an item from the menu to interactively execute the several programs provided.*

ALGORITHMIC NOTATION (showMainMenuProc)

Show the menu items of the program library.

Input

stl1	[Float]
cmd	[Integer>=0]
et2	[Float]
sdata	[List]
odata	[List]

Algorithm showMainMenuAlgo

```
if(cmd == 9) then
    st1 ← time.clock();
    sdata.append("\nMenuitem 9:Show
    Main Menu Session@"
```

```
        +str(time.time())+'\n');
        outputMenu();
        et2 ← time.clock();
        print newline;
        exectime();
    endif.
Output print items in menu from 1 to 17.
Output print execTime.
```

Procedure(Validate and Verify Interactions):

This defintion will define a procedure that helps the reader or student to validate or verify cause – effect models. This is about proof of cause-effect transitions by query language. This uses what i call & ends | middles statement.

Mathematically: & || &.

Symbolically: and or or and.

ALGORITHMIC NOTATION
(validateandVerifyProc)

Show an interactive means to validate and verify a cause-effect model.

Input

stl1	[Float]
cmd	[Integer>=0]
et2	[Float]
sdata	[List]
odata	[List]
selection	[Integer:=1 to 3]

Algorithm validateandVerifyAlgo

```
if(cmd == 10) then
     st1 ← time.clock();
     sdata.append("\nMenuitem 10:
     Validate and Verify Interaction
     Session@"+str(time.time())+'\n');
     cmd ← raw_input('\n__Which
     interactive proof?(\n 1:Cat-Mice
     \n 2:Cat-Human \n 3:Cat-
     Physicale)\n>>');
     odata.append('\n__Which
     interactive proof?(\n 1:Cat-Mice
     \n 2:Cat-Human \n 3:Cat-
     Physicale)\n>>');
     odata.append(str(cmd));
     interactiveproof(int(cmd));
     sdata.append(str(cmd));
     et2 ← time.clock();
     exectime();
endif.
```

Output print fetchdata in database by using selection.

```
Output print execTime.
```

Procedure(Cyber-Simulation): *This is a procedure that defines a stimulation program that teaches prolog-alike syntax of stimulation concern here. There are about 10 stimulae in this program only. Each stimulation is provided after a protime delay or sleep.*

ALGORITHMIC NOTATION (cyberSimulaProc)

Show an interactive means to validate and verify a cause-effect model.

Input

stl1	[Float]
cmd	[Integer>=0]
et2	[Float]
sdata	[List]
odata	[List]
cyberr	[List:={catphy, catmic, cathum}]
selection	[Integer:=1 to 3]
cyberason	[Integer]
simtypes	[List:={cybermetacase, ...}]
stat	[String]

Algorithm cyberSimulaAlgo

if*(cmd == 11)* ***then***

st1 ← time.clock();

cmd ← raw_input("\n___Which cyber-reason? \n(1)catphy\n(2) catmic\n(3) cathum\n>>");

cyberason ← cmd;

cmd ← raw_input("\n___Which systemic cybermodel? \n(1)cybermetacase\n (2)cybermetalang\n (3)cybermetalogic\n>>");

cmd ← int(cmd);

if (cmd== 1) **then**

print '___Systemic CyberModel: Cyber- Metacases.'

queryDatabase(stat)

printFetchData();

endif.

if (cmd == 2) **then**

print '___Systemic CyberModel: Cyber- Metalanguage'

queryDatabase(stat)

printFetchData();

endif.

if (cmd == 3) **then**

print '___Systemic CyberModel: Cyber-Metalogic''

queryDatabase(stat)

printFetchData();

```
 else
   p="___Which simulation
type?\n(1)catsim: Cat
Simulation.\n(2)potsim: Potential
Simulation. \n(3)physim: Physical
Simulation. \n(4)intsim: Intelligence
Simulation.\n(5)symsim: Symbol
Simulation. \n(6)opesim: Operation
Simulation. \n(7)chasim: Chase
Simulation. \n(8)seasim: Search
Simulation.\n(9)Stimulate Defaults\n
 (10)Simulation Info\n>>"
 cmd  ←  raw_input(p);
 queryDatabase(stat);
 stimulate(cybersim);
 endif.
endif.
Output print fetchdata in database by
       using selection variable, cmd.
Output print execTime.
```

Procedure(Transition Semantics): *This procedure defines a program that prints out the semantics of a provided transition function or paths. With the provided, a cause and an effect is concatenated into a new meaning of words. The sense of it is upto the tester to make. It is quite essential to look at the new concepts evolving from it.*

ALGORITHMIC NOTATION
(transitionSemanticsProc)

Show semantics derived from cause-effect path in a ce model by concatenation of cause-effect word.

Input

stl1	[Float]
cmd	[Integer>=0]
et2	[Float]
sdata	[List]
odata	[List]
pri	[Integer:=2]

Algorithm transitionSemanticsAlgo

```
if(cmd == 12) then
     sdata.append("\nMenuitem
     12:Transaction Semantics
     Session@"+ str(time.time())+'\n');
     st1 ← time.clock();
     sdata.append(str(2));
     commandLine(pri);
     et2 ← time.clock();
     print '\n'
     odata.append('\n');
     exectime();
endif;
Output print transition semantics.
Output print execTime.
```

Procedure(Logic Program): *This is the main intelligence program of the interpreter. It is a logic program that stimulates the concept of slot-frame syntax, substitution and unifications. It starts with the particular predicate print, then after a protime-3 sleep, a substituting print then after a protime-2 sleep, a slot-frames print then after a protime-2x sleep and unifications print. It then goes to read-eval-print loop.*

ALGORITHMIC NOTATION (logicProgramProc)

Create a logic procedure that stimulates the concept of slot-frame syntax, subsitution and unifications.

Input

stl1	[Float]
cmd	[Integer>=0]
et2	[Float]
sdata	[List]
odata	[List]

Algorithm `logicProgramAlgo`

```
if(cmd == 13) then
      sdata.append("\nMenuitem
      13:Logic Program@"+
      str(time.time())+'\n');
      st1 ← time.clock();
      sdata.append(str(2));
      print 'Available Frame Predicates'
      cmd ← raw_input('\nWhich logic
      program number?\n>>');
      et2 ← time.clock();
      logicprogram(int(cmd));
      odata.append('Which logic program
      number?');
      exectime();
endif.
Output print frame syntax, substitution,
unification.
Output print execTime.
```

Procedure(Intelligence Builds): *The intelligent builds procedure is made of 6 functions including cybersmart rep (read-eval-print) condition, cyberlearning rep condition, cyberextreme rep condition, cyberquick rep condition, cyberability rep condition and cybersharp rep condition. On*

each rep condition, a particular intelligent build is selected to be intrepreted by first a select query on the cyberintell database and then printing the fetch data in an interactive manner.

ALGORITHMIC NOTATION (buildIntelligenceProc)

Given a rep condition build intelligent functions and print including cyberability, cyberquick, cybersharp, cybersmart, cyber-extreme from a database in an interactive manner.

Input

stl1	[Float]
cmd	[Integer>=0]
et2	[Float]
sdata	[List]
odata	[List]
query	[String:=sql statement]

Algorithm buildIntelligenceAlgo

```
if(cmd == 14) then
    sdata.append("\nMenuitem
    14:Intelligence Builds Session@"+
    str(time.time())+'\n');
    sdata.append('cmd:'+str(cmd));
    st1 ← time.clock();
    print Intelligence Buildings Available;
```

```
cmd  ←  raw_input('\n__Which item
        number?\n>>');
 if (cmd == 1) then
          print 'Cybersmart Buildings'
        queryDatabase(query)
        printFetchData();
 endif.
  if (cmd == 2) then
        print 'Cyberextreme Buildings'
        queryDatabase(query)
        printFetchData();
endif.
 if (cmd == 3) then
        print 'Cyberlearning Buildings'
        queryDatabase(query)
        printFetchData();
endif
 if (cmd == 4) then
        print 'Cyberability Buildings'
        queryDatabase(query)
        printFetchData();
endif.
 if (cmd == 5) then
        print 'Cyberquick Buildings'
        queryDatabase(query)
        printFetchData();
endif.
 if (cmd == 6) then
```

```
                print 'Cybersharp Buildings'
                queryDatabase(query)
                printFetchData();
        else
                print 'Not supported.'
        endif.
Endif.
Output print frame syntax,
substitution, unification.
Output print execTime.
```

Procedure(Show Textfile Session): *This defines a procedure that outputs data just as the screen ouputs during the program testing or execution. There are actually two textfiles mainly cyber-datafile and cyberoutfile. The cyberoutfile is saved on each session of running. It is named with a default-timestamp fashion.*

ALGORITHMIC NOTATION (showTextfileProc)

Create a procedure to prints or outputs of the running program stored in a textfile for testing and debugging purposes.

Input

stl1	[Float]
cmd	[Integer>=0]
et2	[Float]
sdata	[List]
odata	[List]

Algorithm showTextfileAlgo

```
if(cmd == 15) then
    print '\n Textfile Details '
    sdata  ← __loadTextfile;
    for i in range(len(sdata)):
        data  ←  sdata[i]
        print str(data);
 endif.
Output print data in textfile.
Output print execTime.
```

Procedure(Run TextFile Simulation): *This defines a procedure that will use the cyberdatafile to run a textfile simulation to generate an output just like the cyberoutfile saved during a session of running.*

ALGORITHMIC NOTATION

(runTextfileProc)

Create a procedure to use cyberdata file contents as a simulation inputs to repeat already simulated purpose. It is not implemented yet but the we look at the notation.

Input

stl1	[Float]
cmd	[Integer>=0]
et2	[Float]
sdata	[List]
odata	[List]

Algorithm runTextfileAlgo

```
if(cmd == 16) then
     print '\n Textfile Details '
     odata  ← __loadTextfile;
     for i in range(len(odata)):
           data=odata[i]
           stimulate(str(data));
 else
     writeouput(odata);
     writedata(sdata);
     file.close();
     session.close();
 endif.
```

Output stimulate data in textfile or write new data into session file or cyberdata file.

Output print execTime.

Procedure(Press Esc and Enter to quit): *This will define a procedure that quits the whole interpreter and halts a program execution. In this program, the textfiles both cyberoutfile and cyberdatafile files are persisted on quiting the program. Finally, a quit procedure is called to stop the system execution in shell session.*

ALGORITHMIC NOTATION (quitProgramProc)

Create a procedure to store cyberdata file-contents or textfile content as a simulation inputs or output to repeat already simulated purpose. Finally quit the program.

Input

stl1	[Float]
cmd	[Integer>=0]
et2	[Float]
sdata	[List]
odata	[List]

Algorithm quitProgramAlgo

```
if(cmd == 17) then
    st1 ← time.clock();
```

```
        writeoutput(odata)
        writedata(sdata)
        session.close();
        file.close();
        et2 ← time.clock();
        exectime();
        quit();
    endif.

    Output write and close new data into
    session file or cyberdata file and
    finally quit program.

    Output print execTime.
```

1.1.2.2 Algorithm

The algorithm is just like the specific program but will use sub-notation of algorithmic solutions to allow the reader get a look of the earlier defintions-algorithm, program andsoftware. This is similar to algorithmic notation but without declaration of inputs lines, decription lines, output lines and algorithmic notation itself. It is similar to the implementation language, which is Python.

← in the algorithmic notation is replaced by = symbol.

The algorithm allows the reader to :

1. Specify a name of algorithm,
2. Declare variables used in the algorithm,
3. Declares the initialization values for the variables,
4. Begin and end the program interactions or interpretations,
5. Iteration of processes 1-4.

Below is an algorithm:

```
   Algorithm  AlgoName
   if(variable 1 < 2 && variable 2==5)
then
       if('positive'==case) then
    s=Solve-down(+sol);  //+solver
       else
    s=Solve-up(-sol);   //-solution
designer
```

```
        endif
    endif.
```

---

(1) *This program helps one to study the transition functions of a cause-effect(ce) model. Once you know the 9 transition states in ce model of a cyberintel, you can verify a state function of a transition. The algorithm of check transition functions is below:*

```
Algorithm  checkTransitionAlgo

    if(cmd == 1) then
        append(cmd);
        cmdStr=raw_input('Which
            transition?');
        append('Which transition?');
        (cause, effect)=
                  cmdStr.split("=>");
        append(' cause='+cause+'
                    effect='+effect+'  \n');
        for index in range(9)
         valuepair=dictdata[index+1]
         (vcause, veffect) =
       (str(valuepair[0]), valuepair[1])
           if(effct==veffect & cause==vcause)
            then
               s='__Transition Point:
            transit('+str(cause)+',
                  '+str(effect)+')');
           else
```

```
                    s='__Final:No
                transition match for
                '+str(cause)+'=>'+str(effect
                ))
              endif.
        endif.
```

(2) *This program helps one to study the transition points of a cause-effect model. Once you know about 19 or more transition path in ce model of a cyberintel, you can verify a path of model. The algorithm is as shown below:*

```
Algorithm  checkTransitionPoAlgo

      if(cmd == 2) then
          commandLine(cmd);
      endif.
```

Print Transitions: *This algorithm* prints transitions to *study the transition points of a cause-effect model by transversing through the inputted path. Once you know about 19 or more transition paths in ce model of a cyberintel, you can verify a path. The algorithm is as shown below:*

```
      Algorithm  printTransitionAlgo
      if(cmd == 3) then
         commandLine(pri=1);
        endif.
```

Show Stored Views: *There are about 12 views to help one to understand and study the concepts of cyber-intelligence and more.The algorithm is as shown below:*

Algorithm showStoredViewAlgo

```
if(cmd == 4) then
   print views;
   append(views);
   append(cmd);
   selview=raw_input(__Which view
   number?);
   stat='select * from
   '+str(views[selview-1])+';'
   queryDatabase(stat)
   printFetchData();
endif.
```

Show Cause-Effect **Model**: *This will print the entire cause-effect model. There are about 9 transitions in the model with three branches namely cat-mice path, cat-human path and cat-physicale path. The algorithm is as shown below:*

Algorithm *showCEModelAlgo*

```
if(cmd == 5) then
        st1=clock();
```

```
        append(str(cmd));
        showCEModel();
        et2=clock();
        print newline;
        exectime();
endif.
```

Show Natural Hierarchy: *There are about three branches as said above in (5). The program will allow you to select each and then will print the path shape of each cause-effect hierarchy.The algorithm is as shown below:*

Algorithm showNaturalHierAlgo

```
if(cmd == 6) then
        st1=clock();
        cmd =raw_input('__Which natural
        hierarchy?');
        append(str(cmd));
        showHierarchy(int(cmd));
        et2=clock();
        print newline;
        exectime();
endif.
```

***Show Cause-Effect Case**: The three ce model will be printed each as a ce case in non-hierarchical manner. A horizontal view of cause-effect branch is a cause-effect case. The algorithm is as shown below:*

Algorithm showCauseEffectCaseAlgo

```
if(cmd == 7) then
     st1=clock();
     cmd =raw_input('___Which
     behavioral relationship?');
     append(str(cmd));
     showCECase(int(cmd));
     et2=clock();
     print newline;
     exectime();
endif.
```

Book Chapters & Summary: *The chapters of the cyber-intelligence book is printed by as a chapter-by-chapter summary in an interactive manner. The algorithm is as shown below:*

Algorithm showChaptersAlgo

```
if(cmd == 8) then
```

```
                    st1=clock();
                    cmd =raw_input('___Which
                    behavioral relationship?');
                    append(str(cmd));
                    cmd=raw_input('\n___Press Enter for
                    Chapter  1\n>>');
                    print chapter1;
                    cm=raw_input('\n___Press Enter for
                    Chapter  2\n>>');
                    print chapter2;
                    cmd=raw_input('\n___Press Enter for
                    Chapter  3\n>>');
                    print chapter3;
                    cmd=raw_input('\n___Press Enter for
                    Chapter  4\n>>');
                    print chapter4;
                    et2=clock();
                    print newline;
                    exectime();
          endif.
```

Show Main Menu: *This algorithm will print the main menu of the interpreter. This helps one to select an item from the menu to interactively execute the several programs provided.The algorithm is as shown below:*

Algorithm showMainMenuAlgo

```
if(cmd == 9) then
     st1=clock();
     sdata.append("\nMenuitem 9:Show
     Main Menu Session@"
     +str(time.time())+'\n');
     outputMenu();
     et2=clock();
     print newline;
     execime();
endif.
```

Validate and Verify Interactions: *This program helps the reader or student to validate or verify cause–effect models. This is about proof of cause-effect transitions by query language. This uses what i call & ends | middles statement.*

Mathematically: & || &.

Symbolically: and or or and.

The algorithm is as shown below:

Algorithm validateandVerifyAlgo

```
if(cmd == 10) then
     st1=clock();
```

```
cmd=raw_input('\n__Which
interactive proof?(\n 1:Cat-Mice
\n 2:Cat-Human \n 3:Cat-
Physicale)\n>>');

append(str(cmd));

interactiveproof(int(cmd));

et2=clock();

exectime();

endif.
```

Cyber-Simulation: *This is a stimulation algorithm that teaches prolog-alike syntax of stimulation concern here. There are about 10 stimulae in this algorithm only. Each stimulation is provided after a protime delay or sleep. The algorithm is as shown below:*

```
Algorithm  cyberSimulaAlgo

if(cmd == 11) then

 st1=clock();

cmd=raw_input("\n___Which cyber-
        reason? “);

cyberason= cmd;

cmd=raw_input("\n___Which systemic
        cybermodel? ");

cmd = int(cmd);
```

```
 if (cmd== 1) then
        print '___Systemic CyberModel:
        Cyber- Metacases.'
        queryDatabase(stat)
        printFetchData();
 endif.
 if (cmd == 2) then
        print '___Systemic CyberModel:
        Cyber-Metalanguage'
        queryDatabase(stat)
        printFetchData();
endif.
 if (cmd == 3) then
        print '___Systemic CyberModel:
        Cyber-Metalogic''
        queryDatabase(stat)
        printFetchData();
 else
  p="___Which simulation type?"
 cmd=raw_input(p);
queryDatabase(stat);
stimulate(cybersim);
endif.
endif.
```

Transition Semantics: *This algorithm prints out the semantics of a provided transition function or*

paths. With the provided, a cause and an effect is concatenated into a new meaning of words. The sense of it is upto the tester to make. It is quite essential to look at the new concepts evolving from it. The algorithm is as shown below:

Algorithm transitionSemanticsAlgo

```
if(cmd == 12) then
     append("\nMenuitem
     12:Transaction Semantics
     Session@"+ str(time.time())+'\n');
     st1=clock();
     append(str(2));
     commandLine(pri);
     et2=clock();
     print '\n'
     exectime();
endif.
```

Logic Program: *This is the main intelligence algorithm of the interpreter. It is a logic algorithm that stimulates the concept of slot-frame syntax, substitution and unifications. It starts with the particular predicate print, then after a protime-3 sleep, a substituting print then after a protime-2*

sleep, a slot-frames print then after a protime-2x sleep and unifications print. It then goes to read-eval-print loop.The algorithm is as shown below:

Algorithm logicProgramAlgo

```
if(cmd == 13) then
     append("\nMenuitem
     13:Logic Program@"+
     str(time.time())+'\n');
     st1=clock();
     sdata.append(str(2));
     print 'Available Frame Predicates'
     cmd=raw_input('\nWhich logic
     program number?\n>>');
     et2=clock();
     logicprogram(int(cmd));
     append('Which logic program
     number?');
     exectime();
endif.
```

Intelligence Builds: *The intelligent builds algorithm is made of 6 functions including cybersmart rep (read-eval-print) condition, cyberlearning rep condition, cyberextreme rep*

condition, cyberquick rep condition, cyberability rep condition and cybersharp rep condition. On each rep condition, a particular intelligent build is selected to be intrepreted by first a select query on the cyberintell database and then printing the fetch data in an interactive manner. The algorithm is as shown below:

Algorithm buildIntelligenceAlgo

```
if(cmd == 14) then
      append("\nMenuitem 14:Intelligence
             Builds Session@"+
      str(time.time())+'\n');
     append('cmd:'+str(cmd));
     st1=clock();
      print Intelligence Buildings Available;
      cmd=raw_input('\n__Which item
      number?\n>>');
       if (cmd == 1) then
                print 'Cybersmart Buildings'
              queryDatabase(query)
              printFetchData();
       endif.
       if (cmd == 2) then
              print 'Cyberextreme Buildings'
              queryDatabase(query)
```

```
            printFetchData();
        endif.
         if (cmd == 3) then
            print 'Cyberlearning Buildings'
            queryDatabase(query)
            printFetchData();
        endif
         if (cmd == 4) then
            print 'Cyberability Buildings'
            queryDatabase(query)
            printFetchData();
        endif.
         if (cmd == 5) then
            print 'Cyberquick Buildings'
            queryDatabase(query)
            printFetchData();
        endif.
         if (cmd == 6) then
            print 'Cybersharp Buildings'
            queryDatabase(query)
            printFetchData();
         else
            print 'Not supported.'
        endif.
Endif.
```

Show Textfile Session: *This an output data just as the screen ouputs during the program testing or*

execution. There are actually two textfiles mainly cyberdatafile and cyberoutfile. The cyberoutfile is saved on each session of running. It is named with a default-timestamp fashion. The algorithm is as shown below:

Algorithm showTextfileAlgo

```
if(cmd == 15) then
        print '\n Textfile Details '
        sdata =__loadTextfile;
        for i in range(len(sdata)):
                data=sdata[i]
                print str(data);
endif.
```

Run TextFile Simulation: *This will use the cyberdatafile to run a textfile simulation to generate an output just like the cyberoutfile saved during a session of running. The algorithm is as shown below:*

Algorithm *run*TextfileAlgo

```
if(cmd == 16) then
     print '\n Textfile Details '
     odata =__loadTextfile;
     for i in range(len(odata)):
           data=odata[i]
           stimulate(str(data));
 else
     writeouput(odata);
     writedata(sdata);
     file.close();
     session.close();
 endif.
```

Press Esc and Enter to quit: *This quits the whole interpreter and halts a program execution. In this program, the textfiles both cyberoutfile and cyberdatafile files are persisted on quiting the program. Finally, a quit procedure is called to stop the system execution in shell session.The algorithm is as shown below:*

```
Algorithm  quitProgramAlgo
```

```
if(cmd == 17) then
      st1=clock(); #system clock
      writeoutput(odata)
      writedata(sdata)
      session.close();
      file.close();
      et2=clock();
      exectime();
      quit();
endif.
```

1.2 Installing

(1) Unzip the cyberintel.zip file
(2) Open a terminal in the folder by pressing F4.
(3) Type bash installdb.sh or sh installdb.sh
(4) Type your mysql password or change username and password,if defaults -u root and -p are used then no change to the installdb file.
(5) Type python setup.py install | python2 setup.py install | python{version_no} setup.py install
(6) Create a cyberintel program: Type "python" on the terminal

import cyberintelldev;

(7) (4) or Other Run "sh run.sh" at terminal and follow the prompt discussions on the interpreter.
(8) Typical ouput:

```
==================================
WELCOME TO CYBER-INTELLIGENCE
INTERPRETER.
WRITTEN BY Dr. Frank Appiah
YEAR: 2019  MONTH: MAY, 1
Version: 1.0.0
==================================
```

1.3 Summary

This chapter provides a detailed overview of production of programs specifically concerning problem solving phase and implementation phase. Problem solving phase, here concerned a look at the problem definitions of an AI test program and algorithm definitions or designs. This book looked at the definitions about 17 problems of them. Then, both algorithmic notations and algorithm adaptations which are abstracted from the algorithmic notations with few alterations. The implementation phase was touched upon but will be further discussed in Chapter 2.

CHAPTER 2

PYTHON PROGRAMMING OF CYBERINTEL.

The computer can do what you say “do”
It can do anything that you do
Anything, Anything that you do
--Appiah Academy Blog.

CYBERINTEL
AI TEST
INTERPRETER
CODES.

2.0 Cyberintel Implementation Phase

This is the implementation phase of the test program. The actual program code is produced, data abstraction and procedural abstractions are defined. I will look at the python script codes of cyberintel program[6]. Here, over 17 procedure codes along with their pseudocode will be snippet. A pseudocode[2] is a procedure defintion within another procedure definition. The algorithms in Chapter 1 have been translated into Python[1] term for such programlike entities called **def**. Guido van Rossum is the main developer of Python. *Idle* is the integrated development environment of Python programming. A sample procedure definition is as follows:

#program entity 1: Save to file sampleDefOne.py

```
def sampleDef():
1.              print 'This is a sample
                definition one.';
```

#program entity 2: Save to file sampleDefTwo.py

```
def sampleDef(var='Programlike entity'):
1.                              print 'This is a
                        sample definition two.';
2.                      print var;
```

Python Shell
File Edit Shell Debug Options Windows
Python 2.5 (r25:51908, Apr 10 2007, 10:29:19)
[GCC 4.1.2 20070403 (Red Hat 4.1.2-8)] on linux2
Type "copyright", "credits" or "license()" for more information.
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
interface and no data is sent to or received from the Internet.
IDLE 1.2
>>>
Ln: 13

Illustration 1: Idle : Python Shell Programming

In a program containing this procedure definition, identifier *sampleDef* can be used where a declaration can be called for use. The syntax of simple procedure definition is as follows:

- The first part of the definition is the heading:
  def sampleDef():
  def sampleDef(var='Programlike entity'):

1. **This starts with def keyword of Python, then a definition name of the procedure, variable parameter brackets followed by colon.**
2. **This starts with def keyword of Python, then a definition name of the procedure, variable parameter bracket left-side followed by variable parameter name (with initialization) of type, string then a bracket right-side followed by a colon.**

- The second part of the definition is the indented body:
  #program entity 1
  ```
  1. print 'This is a sample definition one.';
  ```
  #program entity 2
  ```
  1. print 'This is a sample definition two.';
  2. print var;
  ```

1. **The program entity 1 is used to refer to the first definition as commented above the definition and brought into the indented code for clarity. The number 1**

is the line number of the script code and there is only one line of code. This simple definition just prints a text,

```
This is a sample definition one.
```

2. **The program entity 2 has only two lines of code. This has one variable parameter passed to the procedure definition. This is *var*, has an intialization value, *Programlike entity* – string type. This definition prints just like 1 but also prints the string value of the var parameter, *Programlike entity*.**
3. **Both ends of the definition ends with a semi-colon (;). This is optional for variable declaration and variable assignments but not problem definition. Most compilers throw indentation error and needs to be checked.**
4. **The body needs to be indented, such that the body text needs to be aligned below the colon line. Just placing the cursor after the colon and entering will give a proper indentation for coding. Forall colons end follows the same concept.**

> **Guido van Rossum[1] believes that using indentation for grouping is extremely elegant and contributes a lot to the clarity of the average Python program.**
>
> **Most people learn to love this feature after a while. - Design FAQ**

- The **#** symbol is used for commenting and documenting a definition or program in Python.

In the comment, programlike entity 1 is saved to sampleDefOne.py whiles that of entity 2 to sampleDefTwo.py. In a procedure call, for example entity 1 calling entity 2 will be declared as follows:

```
def sampleDef():
```

1. `print 'This is a sample definition.';`
2. sampleDef('Entity 1 calling entity 2');

The second entity is now called into the first and now running will show:

```
This is a sample definition one.
This is a sample definition two.
Entity 1 calling entity 2.
```

The call has change the value of var in sampleDef 2 from "`Programlike entity`" to "`Entity 1 calling entity 2.`". Again, if the first calls the second entity but decides not pass a value to the var variable because of the optional implementation of variable parameter. The var is already initialized so can be called without passing a value, it is defaulted to Programlike entity. The sampleDef 1will just be like the sampleDef 2 and that will cause a runtime error or compile error. It looks like a recursive function but is not. It will need to be fix by changing definition names. It follows as:

#program entity 1: Save to file sampleDefOne.py

```
def sampleDefOne():
1.              print 'This is a sample
                definition one.';
```

#program entity 2: Save to file sampleDefTwo.py

```
def sampleDefTwo(var='Programlike entity'):
1.                  print 'This is a sample definition
```

```
                    two.';
2.                  print var;
```

The definition names are now changed. So sampleDefOne will now call sampleDefTwo with no value parameter. It follows as:

#program entity 1: Save to file sampleDefOne.py

```
def sampleDefOne():
1.                print 'This is a sample
                  definition one.';
2.                sampleDefTwo();
```

#program entity 2: Save to file sampleDefTwo.py

```
def sampleDefTwo(var='Programlike entity'):
1.                            print 'This is a
                    sample definition two.';
2.                  print var;
```

Now, calling sampleDefOne in a main definition name will run as:

```
This is a sample  definition one.
This is a  sample definition two.
Programlike entity.
```

There is no errors this time with the minor changes. A third definition with mixed value parameters will follow as:

#program entity 3: Save to file sampleDefThree.py

```
def sampleDefThree(entone,var='Programlike entity'):
1.                      print 'This is a
                         sample definition.';
2.                      print var;
3.                      print 'Programalike
                        Entity '+str(entone);
```

Now, calling sampleDefThree(3) in a main definition name will run as:

```
This is a sample definition.
Programlike entity
Programalike Entity 3
```

Notice, the ***str*** function call to convert **entone** from *int* to *string*.It is imported from a procedure library supported by Python.
Again with the calling of *sampleDefThree* ('Programlike Entity has changed to Programalike

`Entity.', 3)` will run as follows:

```
This is a sample definition.
Programlike Entity has changed to Programalike
Entity.
Programalike Entity 3
```

The defaulted value parameter has changed to the passed value parameter and now we have a different results as output.

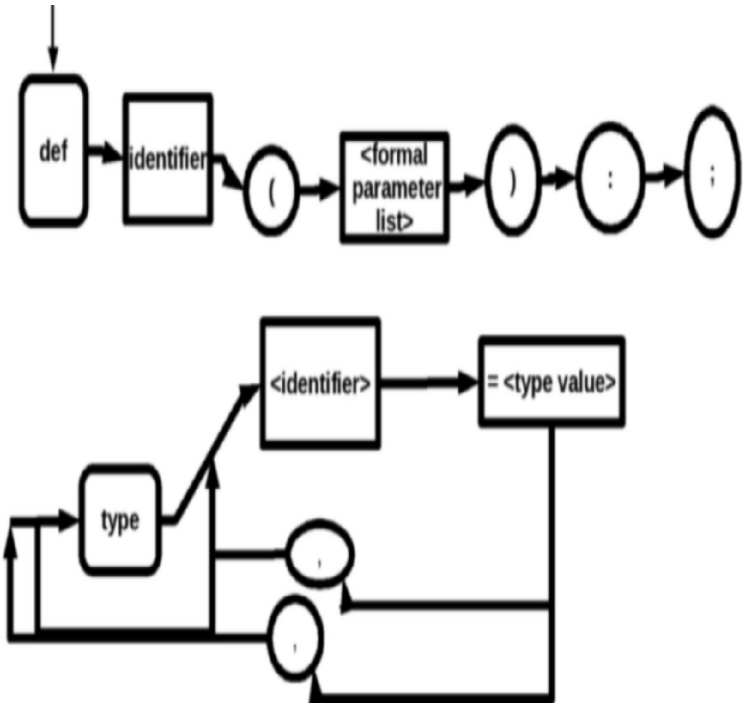


Illustration 2: Syntax for def heading

Lets look at the program code of the production.

First is the check transition functions.

2.1 Program Codes

Check Transition Functions. *This program helps one to study the transition* functions *of a cause-effect(ce) model. Once you know the 9 transition states in ce model of a cyberintel, you can verify a state function of a transition.*

The details of this procedure is below:

```
if(int(cmd)==1):
```

1. `st1=time.clock();`
2. `sdata.append('\  n'+str(cmd));`
3. `cmd=raw_input('\n___Which transition (In lowercases Ex:\n  cat=>potential, cause=>effect)?\n>>')`
4. `odata.append('\n___Which transition (In lowercases Ex:\n  cat=>potential, cause=>effect)?\n>>');`
5. `print '\n'`

```
6.  printLine('____Accepting:$6('+cmd+')')
7.  odata.append('____Accepting:$6('+cmd+')')
8.  sdata.append("\nMenuitem 1:Check Transition
    Functions Session@"+ str(time.time())+'\n');
9.  (cause,
    effect)=cmd.split("=>")cause=str(cause).strip(
    )
10. effect=str(effect).strip()
11. sdata.append('cause='+str(cause)+'effect='+str
    (effect)+'  \n')
12. printLine(str(cause)+'=>'+str(effect))
13. odata.append(str(cause)+'=>'+str(effect))
14. for index in range(9):
    valuepair=dictdata[str(index+1)]
15. (vcause,veffect)=(str(valuepair[0]),valuepair[
    1])
16. printLine('__Checking transition
    state'+str(index+1)+':'+str(vcause)+'=>'+str(v
    effect))
17. odata.append('__Checking transition state
    '+str(index+1)+':'+str(vcause)+'=>'+str(veffec
    t))
18. time.sleep(protime);
19.  if((str(cause).strip()==(str(vcause).strip())
    ) and
    (str(effect).strip()==str(veffect).strip())):
20. printLine('__Transition Point:
    transit('+str(cause)+','+str(effect)+')');
21. odata.append('__Transition Point
    transit('+str(cause)+', '+str(effect)+')');
22.  break;
23.   else:
24.        printLine('__Final:No transition match
```

```
        for '+str(cause)+'=>'+str(effect))
```

```
25.       odata.append('__Final:No transition
   match for '+str(cause)+'=>'+str(effect))
26. print '\n'
27. et2=time.clock();
28. exectime();
```

Pseudocode Listing:

printLine pseudocode:

Script Listing (Lines 5,6,12,16,20,26):

```
def printLine(txt):
      print(txt);
```

Check Transition Points. *This program helps one to study the transition points of a cause-effect model. Once you know about 19 or more transition path in ce model of a cyberintel, you can verify a path of model.*

The details of this procedure is below:

---

elif(int(cmd)==2):

```
1.                st1=time.clock();
2.                sdata.append("\nMenuitem 2:Check
3.                      Transition Points  Session@"
4.                      +str(time.time())+'\n');
5.                sdata.append(str(cmd));
6.                commandLine();
7.                et2=time.clock();
8.                print '\n'
9.                exectime();
```

Pseudocode Listing:

commandLine pseudocode:

Script Listing:

```
def commandLine(pri=0):
1.    cmd=raw_input('\n___Which transition 2.
      path(In lowercases Ex:\n cat=>potential,
      cause=>effect)?\n\n>>')
2.    odata.append('\n___Which transition
      path(In lowercases Ex:\n cat=>potential,
      cause=>effect)?\n\n>>');
3.    odata.append(cmd);
4.    sdata.append('cmd:'+cmd);
5.    print '\n'
6.    p='__Accepting path:'+cmd
7.    print p;
8.    odata.append('__Accepting path:'+cmd);
      (cause, effect)=cmd.split("=>")
```

```
9.    cause=str(cause).strip()
10.   effect=str(effect).strip()
11.   printLine(str(cause)+'=>'+str(effect))
12.   odata.append(str(cause)+'=>'
      +str(effect))
13.   cmd=raw_input('\n___Which Interaction?
      \n (a)Cat-Mice Interaction\n (b)Cat-
      Human Interaction\n (c)Cat-Physicale
      Interaction\n\n>>>')
14.   odata.append('\n___Which Interaction? \n
      (a)Cat-Mice Interaction\n (b)Cat-Human
      Interaction\n (c)Cat-Physicale
      Interaction\n\n>>>');
15.   cmd=str(cmd).strip()
16.   odata.append(cmd);
17.   sdata.append('cmd:'+cmd);
18.   if(pri==0):
19.         if(str(cmd)==str('a')):
20.               checkTransition(cause,
                  effect, dictdata, 'a');
21.         elif(str(cmd)==str('b')):
22.               checkTransition(cause,
                  effect, dictdata, 'b');
23.         elif(str(cmd)==str('c')):
24.               checkTransition(cause,
                  effect, dictdata, 'c');
25.   elif(pri==1):
26.         if(str(cmd)==str('a')):
```

```
27.               checkTransition(cause,
                  effect, dictdata, 'a',
                  pri=1);
28.         elif(str(cmd)==str('b')):
29.               checkTransition(cause,
                  effect, dictdata, 'b',
                  pri=1);
30.         elif(str(cmd)==str('c')):
31.               checkTransition(cause,
                  effect, dictdata, 'c',
                  pri=1);
32.         elif(pri==2):
33.         if(str(cmd)==str('a')):
34.               checkTransition(cause,
                  effect, dictdata, 'a',
                  pri=2);
35.         elif(str(cmd)==str('b')):
36.               checkTransition(cause,
                  effect, dictdata, 'b',
                  pri=2);
37.         elif(str(cmd)==str('c')):
38.               checkTransition(cause,
                  effect, dictdata, 'c',
                  pri=2);
```

checkTransition pseudocode:

Script Listing (Lines 20,22,24,27, 29,31,34, 36,38):

```
Refer to appendix for pseudocode details.
```

execTime pseudocode:

Script Listing (Line 5):

```
def execTime():
1.    print '\nStart Time='+str(st1)+' End
2.    Time='+str(et2)+' Execution
3.    Time:'+str(et2-st1);
4.    print 'CPU time:'+str(time.clock());
5.    odata.append('\nStart Time='+str(st1)+'
6.    End Time='+str(et2)+' Execution
7.    Time:'+str(et2-st1));
8.    odata.append('CPU
9.    time:'+str(time.clock()));
```

Print Transitions. *This program prints transitions to study the transition points of a cause-effect model by transversing through the inputted path. Once you know about 19 or more transition paths in ce model of a cyberintel, you can verify a path.*

The details of this program is below:

```
elif(int(cmd)==3):
1.                st1=time.clock();
```

```
2.                sdata.append("\nMenuitem 3:Print
                        Transition Session@"
                        +str(time.time())+'\n');
3.                commandLine(pri=1)
4.                et2=time.clock();
5.                print '\n'
6.                exectime();
```

---

execTime pseudocode:

Script Listing (Line 6):

```
def execTime():
#Check above for details or refer to complete
script code in appendix.
```

commandLine pseudocode:

Script Listing (Line 3):

```
def commandLine(pri=1):
#Check above for details or refer to complete
script code in appendix.
```

Show Stored Views.*There are about 12 views to help one to understand and study the concepts of cyber-intelligence and more.*

The details of this program is below:

```
elif(int(cmd)==4):
1.                st1=time.clock();
2.                sdata.append("\nMenuitem 4:Show
                  Stored Views Session@"
                  +str(time.time())+'\n');
3.             views=['cyberdesignview','humceview',
                  'inteceview','miceview',
                  'opeceview','phycatview',
                  'phyceview', 'potceview',
4.                'seaceview','symceview',
                  'psychocatview','chaceview']
5.                p='\n____Selection: Stored Views'
6.                print p
7.                odata.append(p);
8.                p='\n\nView:No
                  Description'
9.                print p
10.               odata.append(p);
11.               p='----------------------------'
12.               print p
13.               odata.append(p);
14.               p='cyberdesignview:1    Cyberdesign
                  Views'
15.               print p
16.               odata.append(p);
17.               p='humceview:2          Human
                  Interaction Views.'
18.               print p
                  odata.append(p);
19.               p='inteceview:3
                  Intelligence Interaction Views.'
```

```
20.             print p
21.             odata.append(p);
22.             p='miceview:4           Mice
                Interaction      Views'
23.             print p
24.             odata.append(p);
25.             p='opeceview:5          Potential
                Interaction Views'
26.             print p
27.             odata.append(p);
28.             p='phycatview:6         Physical
                Cat              Views.'
29.             print p
30.             odata.append(p);
31.             p='phyceview:7          Physical
                Interaction Views.'
32.             print p
33.             odata.append(p);
34.             p='potceview:8          Potential
                Interaction Views.'
35.             print p
36.             odata.append(p);
37.             p='seaceview:9          Search
                Interaction Views.'
38.             print p
39.             odata.append(p);
40.             p='symceview:10         Symbol
                Interaction Views.'
41.             print p
42.             odata.append(p);
43.             p='psychocatview:11     Pyscho-cat
```

```
                Views'
44.             print p
45.             odata.append(p);
46.             p='chaceview:12         Chase
                      Interaction Views.'
47.             print p
48.             odata.append(p);


49.             cmd=raw_input('\n__Which view
                number? \n>>')
50.             odata.append('\n__Which view
                number?\n>>')
51.             selview=int(cmd)
52              sdata.append(str(cmd));
53.             odata.append(str(cmd));
54.             p='\nCyberactions: Displaying View
                '+str(selview)+':'+str(views
                [selview-1])
55.             print p
56.             odata.append(p);
57.             p='\n'
58.             print p
59.             odata.append(p);
60.             stat='select * from
                '+str(views[selview-1])+';'
61.             sdata.append(stat);
62.             if(selview>12 or selview<1):
63.                   p='___View is not
                      available.';
64.                   print p
65.                   odata.append(p);
```

```
66.             queryDatabase(stat)
67.             printFetchData();
68.             et2=time.clock();
69.             p='\n';
70.             print p
71.             odata.append(p);
72.             exectime();
```

---

queryDatabase pseudocode:

Script Listing (Line 66):

```
def queryDatabase(stat):
#Refer to complete script code in appendix.
```

printFetchData pseudocode:

Script Listing (Line 67):

```
def printFetchData():
#Refer to complete script code in appendix.
```

exectime pseudocode:

Script Listing (Line 72):

```
def exectime():
#Refer to complete script code in appendix.
```

Show Cause-Effect Model. *This will print the entire cause-effect model. There are about 9 transitions in the model with three branches*

namely cat-mice path, cat-human path and cat-physicale path.

The details of this program is below:

```
elif(int(cmd)==5):
1.                st1=time.clock();
2.                sdata.append("\nMenuitem 5:Show
                  Cause-Effect Model Session@"
                        +str(time.time())+'\n');
3.                sdata.append(str(cmd));
4.                odata.append(str(cmd));
5.                showCEModel();
6.                et2=time.clock();
7.                print '\n'
8.                exectime();
```

showCEModel pseudocode:

Script Listing (Line 5):

```
def showCEModel():
#Refer to complete script code in appendix.
```

exectime pseudocode:

Script Listing (Line 8):

```
def exectime():
#Refer to complete script code in appendix.
```

Show Natural Hierarchy. *There are about three branches as said above in (5). The program will allow you to select each and then will print the path shape of each cause-effect hierarchy.*

The details of this program is below:

```
elif(int(cmd)==6):
1.              st1=time.clock();
2.              sdata.append("\nMenuitem 6:Show
                Natural Hierarchy Session@"
                +str(time.time())+'\n');
3.              cmd=raw_input('\n___Which
                transition hierarchy\n (1:Cat-Mice
                \n 2: Cat-Human \n 3:Cat-
                Physicale\n)?\n>>');
4.              odata.append('\n___Which
                transition hierarchy\n (1:Cat-Mice
                \n 2: Cat-Human \n 3:Cat-
                Physicale\n)?\n>>');
5.              odata.append(str(cmd));
6.              print '\n'
7.              p='Accepting: $6('+cmd+' )'
8.              print p;
9.              odata.append(p);
10.             showHierarchy(int(cmd));
```

```
11.                sdata.append(str(cmd));
12.                et2=time.clock();
13.                exectime();
```

---

showHierarchy pseudocode:

Script Listing (Line 10):

```
def showHierarchy(int):
#Refer to complete script code in appendix.
```

Show Cause-Effect Case. *The three ce model will be printed each as a ce case in non-hierarchical manner. A horizontal view of cause-effect branch is a cause-effect case.*

The details of this program is below:

```
elif(int(cmd)==7):
1.                st1=time.clock();
2.                sdata.append("\nMenuitem 7:Show
                  Cause-Effect Case Session@"
                  +str(time.time())+'\n');
3.                cmd=raw_input('\n___Which
                  behavioral relationship?(\n 1:Cat-
                  Mice \n 2:Cat-Human \n 3:Cat-
                  Physicale\n)\n>>');
4.                odata.append('\n___Which
                  behavioral relationship?(\n 1:Cat-
                  Mice \n 2:Cat-Human \n 3:Cat-
```

```
                Physicale\n)\n>>');
5.              odata.append(str(cmd));
6.              print '\n'
7.              showCECase(int(cmd));
8.              sdata.append(str(cmd));
9.              et2=time.clock();
10.             exectime();
```

---

showCECase pseudocode:

Script Listing (Line 7):

```
def showCECase(int):
#Refer to complete script code in appendix.
```

Book Chapters & Summary. *The chapters of the cyber-intelligence book is printed by as a chapter-by-chapter summary in an interactive manner.*

The details of this program is below:

```
elif(int(cmd)==8):
1.              st1=time.clock();
2.              sdata.append("\nMenuitem 8:Print
                Book Chapters & Summary Session@"
                +str(time.time())+'\n');
```

```
3.              p='Selection: Book Summary &
                Chapters\n'
4.              print p
5.              odata.append(p)
6.              cmd=raw_input('\n_____Press Enter
                for Chapter 1\n>>');
7.              odata.append('\n_____Press Enter
                for Chapter 1\n>>')
8.              print chapter1;
9.              odata.append(chapter1);
10.             cmd=raw_input('\n____Press Enter
                for Chapter 2\n>>');
11.             odata.append('\n____Press Enter
                for Chapter 2\n>>');
12.             print chapter2;
13.             odata.append(chapter2);
14.             cmd=raw_input('\n___Press Enter
                for Chapter 3\n>>');
15.             odata.append('\n___Press Enter for
                Chapter 3\n>>');
16.             print chapter3;
17.             odata.append(chapter3);
18.             cmd=raw_input('\n__Press Enter for
                Chapter 4\n>>');
19.             odata.append('\n__Press Enter for
                Chapter 4\n>>');
20.             print chapter4;
21.             odata.append(chapter4);
22.             et2=time.clock();
23.             exectime()
```

Show Main Menu. *This program will print the main menu of the interpreter. This helps one to select an item from the menu to interactively execute the several programs provided.*

The details of this program is below:

```
elif(int(cmd)==9):
1.                st1=time.clock();
2.                sdata.append("\nMenuitem 9:Show 3
                  Main  Menu Session@"
                  +str(time.time())+'\n');
3.                outputMenu();
4.                et2=time.clock();
5.                print '\n'
6.                exectime();
```

outputMenu pseudocode:

Script Listing (Line 3):

```
def outputMenu():
#Refer to complete script code in appendix.
```

exectime pseudocode:

Script Listing (Line 6):

```
def exectime():
#Refer to complete script code in appendix.
```

Validate and Verify Interactions.*This program helps the reader or student to validate or verify cause – effect models. This is about proof of cause-effect transitions by query language. This uses what i call & ends | middles statement.*

Mathematically: & || &.

Symbolically: and or or and.

The details of this program is below:

```
elif(int(cmd)==10):
1.              st1=time.clock();
2.              sdata.append("\nMenuitem 10:
                Validate and Verify Interaction
                Session@"+str(time.time())+'\n');
3.              cmd=raw_input('\n__Which
                interactive proof?(\n 1:Cat-Mice
                \n 2:Cat-Human \n 3:Cat-
                Physicale)\n>>');
4.              odata.append('\n__Which
                interactive proof?(\n 1:Cat-Mice
                \n 2:Cat-Human \n 3:Cat-
                Physicale)\n>>');
5.              odata.append(str(cmd));
6.              print '\n'
```

```
7.                interactiveproof(int(cmd));
8.                sdata.append(str(cmd));
9.                et2=time.clock();
10.               exectime();
```

---

interactiveproof pseudocode:

Script Listing (Line 7):

```
def interactiveproof(int):
#Refer to complete script code in appendix.
```

exectime pseudocode:

Script Listing (Line 10):

```
def exectime():
#Refer to complete script code in appendix.
```

Cyber-Simulation. *This is a stimulation program that teaches prolog-alike syntax of stimulation concern here. There are about 10 stimulae in this program only. Each stimulation is provided after a protime delay or sleep.*

The details of this program is below:

```
elif(int(cmd)==11):
1.                st1=time.clock();
2.                sdata.append("\nMenuitem 11:Cyber-
```

```
                Simulation Session@"+
                str(time.time())+'\n');
                simtypes=['cybermetacase',
                'cybermetalang','cyberason',
              'cybersimulation',
               'cybersimbehavior',
                'cybermetalogic'];
3.              cyberr=['catphy', 'catmic',
                'cathum']
4.              cmd=raw_input("\n___Which cyber-
                reason? \n(1)catphy\n(2)
                catmic\n(3)cathum\n>>");
5.              odata.append("\n___Which cyber-
                reason? \n(1)catphy\n
                (2)catmic\n(3)cathum\n>>");
6.              cyberason=int(cmd);
                odata.append(str(cyberason));
7.              cmd=raw_input("\n___Which systemic
                cybermodel? \n (1)cybermetacase
                \n(2)cybermetalang\n
                (3)cybermetalogic\n>>");
8.              odata.append("\n___Which systemic
                cybermodel?\n
                (1)cybermetacase\n(2)
                cybermetalang\n
                (3)cybermetalogic\n>>");
9.              cmd=int(cmd);
10.             sdata.append(str(cmd));
11.             odata.append(str(cmd));
12.             if(cmd==1):
13.                   p='\n___Systemic CyberModel:
                      Cyber-Metacases.';
14.                   print p
15.                   odata.append(p);
16.                   stat="select cybercase from
```

```
                    cybermetacase where
17.                 cyberreason='"+
                  str(cyberr[cyberason-1])+"';";
18.                 sdata.append('query:'+stat);
19.                 odata.append('query:'+stat);
20.                 queryDatabase(stat);
21.                 printFetchData();
22              elif(cmd==2):
23.                 p='\n___Systemic CyberModel:
                    Cyber-Metalanguage';
24.                 print p
25.                 odata.append(p);
26.                 query="select cyberlang from
                    cybermetalang where
26.                 fid="+str(cyberason)+";";
27.                sdata.append('query:'+query);
28.               odata.append('query:'+query);
29.                 queryDatabase(query);
30.                 printFetchData();
31.             elif(cmd==3):
32.                 p='\n___Systemic CyberModel:
                    Cyber-Metalogic';
33.                 print p
34.                 odata.append(p);
35.                 query="select cybereason
                    from cybermetalogic
                    where fid="+str(cyberason)
                    +" order by mid;";
36.                sdata.append('query:'+query);
37.                 odata.append(query);
```

```
38.                     queryDatabase(query);
                        printFetchData();
39.               else: p='\n___Not supported in
                        systemic modeling';
40.               print p
41.               odata.append(p);
42.               p="\n___Which simulation
                  type?\n(1)catsim:
                   Cat Simulation.\n(2)potsim:
                   Potential Simulation.\n(3)physim:
                   Physical Simulation. \n(4)intsim:
                Intelligence Simulation.\n(5)symsim:
                   Symbol Simulation. \n(6)opesim:
                  Operation Simulation.\n(7)chasim:
                  Chase Simulation.\n(8)seasim:
                  Search Simulation.\n(9)Stimulate
                  Defaults\n(10)Simulation
                  Info\n>>";
43.               cmd=raw_input(p);
44.               odata.append(p);
45.               odata.append(str(cmd));
46.               sdata.append("simtype:"+str(cmd));
47.               cybersim=int(cmd);

48.               sdata.append('cybersim:'
                  +str(cybersim));
49.               query=simviewtrife;
50.               query=query+" where
                  fid='"+str(cyberason)+"' and
                  simv='"+str(cybersim)+"' ;";
51.               sdata.append('query:'+query);
52.               odata.append(query);
53.               queryDatabase(query);
```

```
54.             stimulate(cybersim);
55.             et2=time.clock();
56.             print '';
57.             exectime();
```

---

queryDatabase *pseudocode:*

Script Listing (Lines 20, 29, 53):

```
def queryDatabase(int):
#Refer to complete script code in appendix.
```

printFetchData *pseudocode:*

Script Listing (Lines 21, 30):

```
def printFetchData(int):
#Refer to complete script code in appendix.
```

stimulate *pseudocode:*

Script Listing (Line 54):

```
def stimulate(int):
#Refer to complete script code in appendix.
```

Transition Semantics. *This program prints out the semantics of a provided transition function or paths. With the provided, a cause and an effect is concatenated into a new meaning of words. The sense of it is upto the tester to make. It is quite*

essential to look at the new concepts evolving from it.

The details of this program is below:

```
      elif(int(cmd)==12):
1.                sdata.append("\nMenuitem
                  12:Transition Semantics Session@"+
                  str(time.time())+'\n');
2.                st1=time.clock();
3.                sdata.append(str(2));
4.                commandLine(pri=2)
5.                et2=time.clock();
6.                print '\n'
7.                odata.append('\n');
8.                exectime();
```

commandLine *pseudocode:*

Script Listing (Line 4):

```
def commandLine(int):
#Refer to complete script code in appendix.
```

Logic Program. *This is the main intelligence program of the interpreter. It is a logic program that stimulates the concept of slot-frame syntax, substitution and unifications. It starts with the*

particular predicate print, then after a protime-3 sleep, a substituting print then after a protime-2 sleep, a slot-frames print then after a protime-2x sleep and unifications print. It then goes to read-eval-print loop.

The details of this program is below:

```
      elif(int(cmd)==13):
1.                sdata.append("\nMenuitem 13:Logic
                  Program Session@"+
                  str(time.time())+'\n');
2.                st1=time.clock();
3.                p='Logic Program\n';
4.                print p
5.                odata.append(p);
6.                cmd=raw_input('Enter to start
                  logic program>>');
7.                odata.append('Enter to start logic
                  program>>')
8.                p='\n____Frame Predicates
                  Available____'
9.                print p
10.               odata.append(p);
11.               p='1. ceiling frame(s).'
12.               print p
13.               odata.append(p);
14.               p='2. create frame.'
```

```
15.             print p
16.             odata.append(p);
17.             p='3. chase frame.'
18.             print p
19.             odata.append(p);
20.             p='4. attack frames.'
21.             print p
22.             odata.append(p);
23.             p='5. disentangle frame.'
24.             print p
25.             odata.append(p);
26.             p='6. escape frame.'
27.             print p
28.             odata.append(p);
29.             p='7. leave frame.'
30.             print p
31.             odata.append(p);
32.             cmd=raw_input('\nWhich logic
                program number?\n>>');
33.             odata.append('\nWhich logic
                program number?\n>>');
34.             sdata.append('cmd:'+str(cmd));
35.             logicprogram(int(cmd));
36.             et2=time.clock();
37.             exectime();
```

---

logicprogram *pseudocode:*

Script Listing (Line 35):

```
def logicprogram(int):
#Refer to complete script code in appendix.
```

Intelligence Builds. *The intelligent builds program is made of 6 functions including cybersmart rep (read-eval-print) condition, cyberlearning rep condition, cyberextreme rep condition, cyberquick rep condition, cyberability rep condition and cybersharp rep condition. On each rep condition, a particular intelligent build is selected to be intrepreted by first a select query on the cyberintell database and then printing the fetch data in an interactive manner.*

The details of this program is below:

```
elif(int(cmd)==14):
1.              sdata.append("\nMenuitem
                14:Intelligence Builds Session@"+
                str(time.time())+'\n');
2.              sdata.append('cmd:'+str(cmd));
3.              st1=time.clock();
4.              p='\n____Intelligence Buildings';
5.              print p
6.              odata.append(p);
7.              p=''
```

```
                print p
8.              odata.append(p);
9.              p='1: cybersmart';
10.             print p
11.             odata.append(p);
12.             p='2: cyberextreme';
13.             print p
14.             odata.append(p);
15.             p='3: cyberlearning';
16.             print p
17.             odata.append(p);
18.             p='4: cyberability';
19.             print p
20.             odata.append(p);
2121.           p='5: cyberquick';
22.             print p
23.             odata.append(p);
24.             p='6: cybersharp';
25.             print p
26.             odata.append(p);
27.             cmd=raw_input('\n__Which item
                number?\n>>');
28.             odata.append('\n__Which item
                number?\n>>');
29.             buid=int(cmd);
30.             sdata.append(str(cmd));
31.             cmd=buid;
32.             query=""
33.             if(cmd==1):
```

```
34.                     p='Cybersmart Buildings'
35.                     print p
36.                     odata.append(p)
37.                     query="select cybersmart
                        from intelligentbuilds;";
38.                     sdata.append('\n'+query)
39.                     queryDatabase(query);
40.                     printFetchData();
41.             elif(cmd==2):
42.                     p='Cyberextreme Buildings'
43.                     print p
44.                     odata.append(p);
45.                     query="select cyberextreme
                        from intelligentbuilds;";
46.                     sdata.append('\n'+query);
47.                     queryDatabase(query);
48.                     printFetchData();
49.             elif(cmd==3):
50.                     p='Cyberlearning Buildings'
51.                     print p
52.                     odata.append(p);
53.                     query="select cyberlearning
                        from intelligentbuilds;";
54.                     sdata.append(query);
55.                     queryDatabase(query);
56.                     printFetchData();
57.             elif(cmd==4):
58.                     p='Cyberability Buildings'
59.                     print p
```

```
60.                     odata.append(p);
61.                     query="select cyberability
                        from intelligentbuilds;";
62.                     sdata.append(query);
63.                     queryDatabase(query);
64.                     printFetchData();
65.               elif(cmd==5):
66.                     p='Cyberquick Buildings'
67.                     print p
68.                     odata.append(p);
69.                     query="select cyberquick
                        from intelligentbuilds;";
70.                     sdata.append(query);
71.                     queryDatabase(query);
72.                     printFetchData();
73.               elif(cmd==6):
74.                     p='Cybersharp Buildings'
75.                     print p
76.                     odata.append(p);
77.                     query="select cybersharp
                        from intelligentbuilds;";
78.                     sdata.append(p);
79.                     queryDatabase(query);
80.                     printFetchData();
81.               else:
82.                     print 'Not supported.'
83.               et2=time.clock();
84.               exectime();
```

queryDatabase *pseudocode:*

Script Listing (Lines 39, 47, 55, 63, 71, 79):

```
def queryDatabase(statement):
#Refer to complete script code in appendix.
```

printFetchData *pseudocode:*

Script Listing (Line 40, 48, 56, 64, 72, 80):

```
def printFetchData(int):
#Refer to complete script code in appendix.
```

Show Textfile Session. *This an output data just as the screen ouputs during the program testing or execution. There are actually two textfiles mainly cyberdatafile and cyberoutfile. The cyberoutfile is saved on each session of running. It is named with a default-timestamp fashion.*

The details of this program is below:

12

```
      elif(int(cmd)==15):
1.                print '\n Textfile Details '
2.                print '====================\n'
3.                for i in range(len(sdata)):
4.                      data=sdata[i]
5.                      print str(data);
```

Run TextFile Simulation. *This will use the cyberdatafile to run a textfile simulation to generate an output just like the cyberoutfile saved during a session of running.*

The details of this program is below:

---

elif(int(cmd)==16):

```
1.          p='Selected: Textfile Simulation'
2.          print p
3.          odata.append(p);
4.          print 'not implemented'
```

Press Esc and Enter to quit. *This quits the whole interpreter and halts a program execution. In this program, the textfiles both cyberoutfile and cyberdatafile files are persisted on quiting the program. Finally, a quit procedure is called to stop the system execution in shell session.*

The details of this program is below:

```
      else:
1.                writeouput(odata);
2.                writedata(sdata);
3.                file.close();
4.                session.close();
5.                print '\n'
```

---

writeouput *pseudocode:*

Script Listing (Line 1):

```
def writeoutput(List):
#Refer to complete script code in appendix.
```

writedata *pseudocode:*

Script Listing (Line 2):

```
def writedata(List):
#Refer to complete script code in appendix.
```

2.2 Testing and Debugging

This section describes how to test the AI progams to validate and accept functions implemented in the implementation phase- actual program codes. The test script code file with the implemented programs is *cyberintelldev* and that of the debug

script code programs is *cyberintelldebug*.

2.2.1 Program Testing

There are two main groupings of testing namely[3]:

- Black-box Testing
- Glass-box Testing

Black-box Testing is the approach of testing focusing on inputs, outputs and principle functions of software module. *Glass-box Testing* is the approach to testing that looks into the structure of code for a software module. Black-box testing is also called *functional testing* and Glass-box testing is also called *structural testing*.
A taxonomy of software testing embraces the following categories:

- Specification oriented.
- Implementation oriented.
- Error oriented.

Specification-oriented testing is an orientation of

testing developed by test data from documents and its purpose is to specify system behavior. Specification oriented testing assumes functional view of software and sometimes called also as black box testing.

Implementation-oriented testing is a class of testing guided by test data derived by information (text/data) from the implementation. Each execution follows a specific program path.

Error-oriented testing is a class of testing that emerges by error data derived by presence of of errors in the programming process.

There are two main forms of testing namely:

1. Static Form and
2. Dynamic Form. (See Table Below)

The problem solving phase of the program production deals with problem defintions and algorithmic design- algorithmic notation and algorithm. I will use the forms of testing tabulated as shown below to describe the testing done so far.

	Black Box (Functional)	***Glass Box (Structural)***
Dynamic	Random testing. Domain testing. Cause-Effect graphing.	Computation testing. Domain testing. Path-based testing. Data generation. Mutation analysis.
Static	Specification proving	Code walkthroughs Inspections Program proving Symbiolic execution Anomaly analysis.

Table 1: Taxonomy of Forms of Testing [Source: James Peters & Witold Pedcrycz, Cf. 3]

CYBERINTEL FORMS OF TESTING

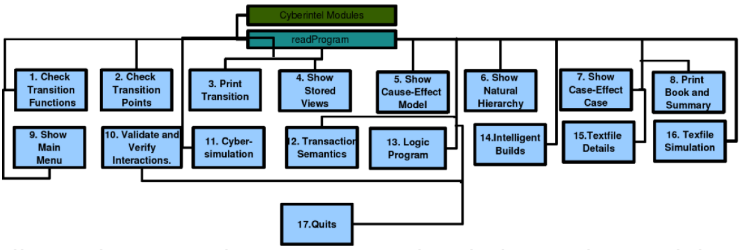


Illustration 3: readProgram: Functional Abstraction Module

The hierarchical diagram above is an integration testing model for readProgram module in cyberintel program. The 17 functional modules of readprogram are independent of each other. These are what the customer will need or required.

	Black Box (Functional)	***Glass Box (Structural)***
Dynamic	Random testing. 1. Functional execution by selective items(1 to 17) at random displaying outputs from each selection. Domain testing. 1. Functional testing of switch state-process by input menu selection of execution program.	Data generation. 1. Data generated from debug process in textfile. 2. Data generated from inform-ation logging process in textfile.
Static	Specification proving 1. Problem defintions are made up of 17 specification texts. (Chapter 1)	Code walkthroughs 1. Algorithmic notation of 17 codes of cyberintel. (chapter 1) 2. Algorithm design of cyberintel programs.(chap ter 1) 3. Implementation codes of cyberintel programs in Chapter 2. Program proving 1. Program codes snippets of cyberintel programs

	Black Box (Functional)	Glass Box (Structural)
		2. (Chapter 2) Pseudocode listings in graphical forms in Appendix B.

Table 2: Cyberintel Forms of Testing

Program Execution

Method: UnCompiled Script

At folder directory, press F4 to start the linux shell itself. Then type ass shown below:

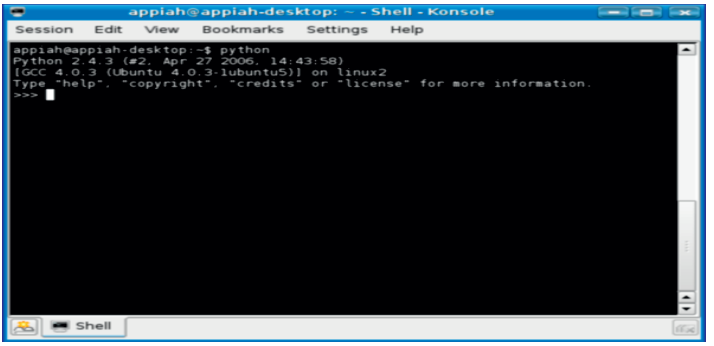
```
python cyberintelldev
```

The program should be running by now. Using the keyboard one can interactively execute the program from the input menu-items as shown at display.

Method: Compiled Script

Follow the installation instructions as shown in Chapter 1. By invoking the python shell, you can

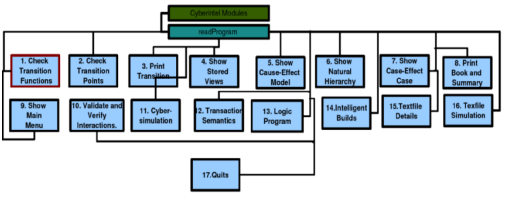
run the complied script from the shell as shown


appiah@appiah-desktop: ~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
appiah@appiah-desktop:~$ python
Python 2.4.3 (#2, Apr 27 2006, 14:43:58)
[GCC 4.0.3 (Ubuntu 4.0.3-1ubuntu5)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
Shell

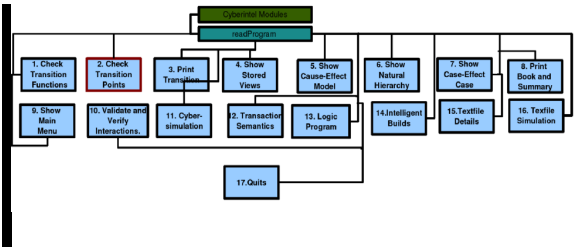
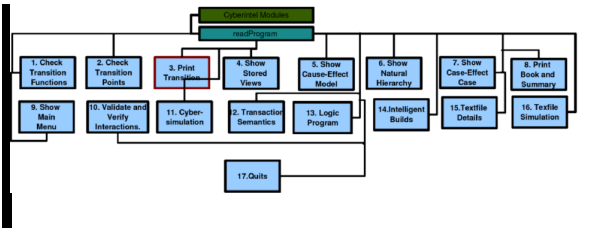
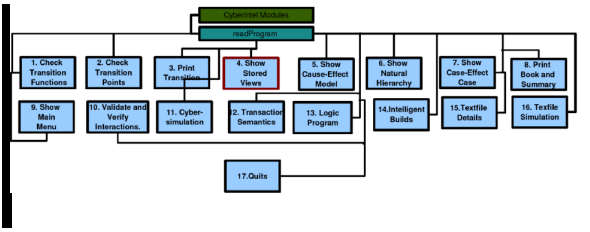
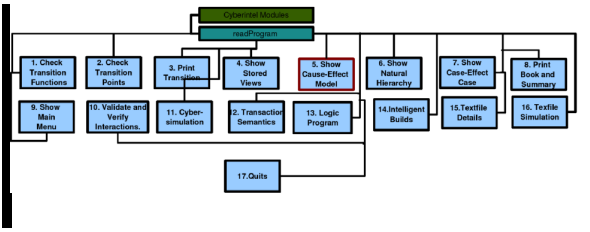
above. By importing cyberintelldev at the prompt (>>>) as shown:

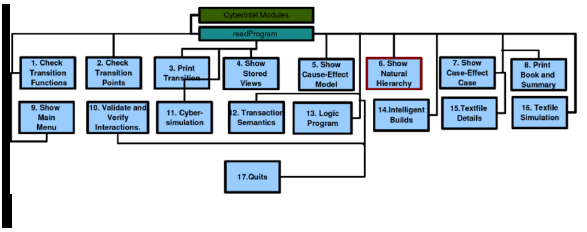
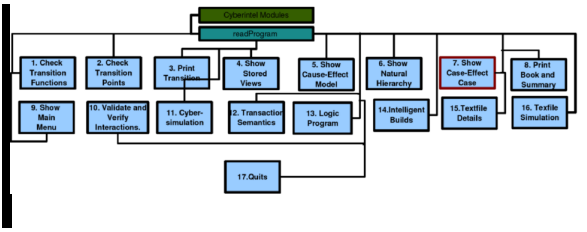
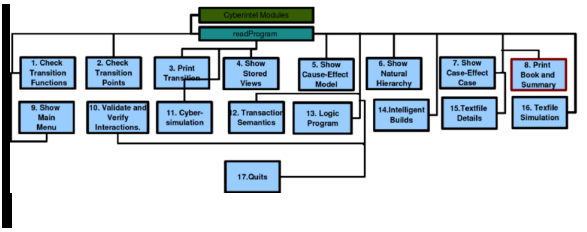
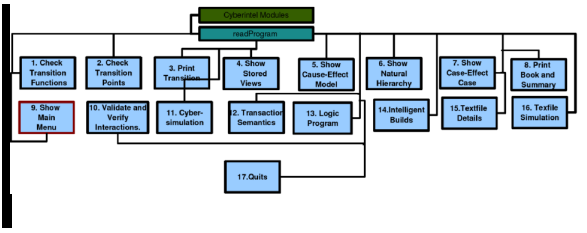
```
import cyberintelldev
```

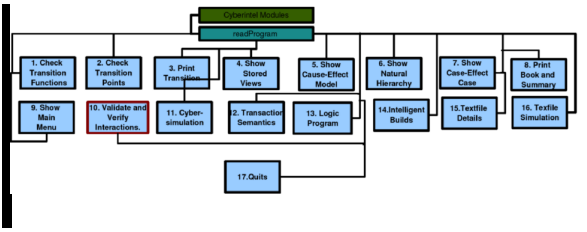
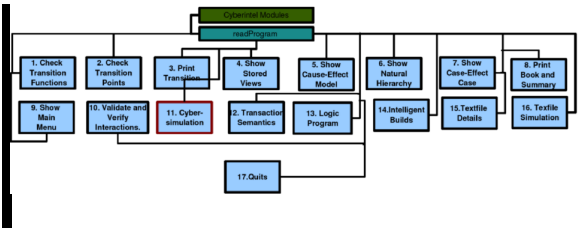
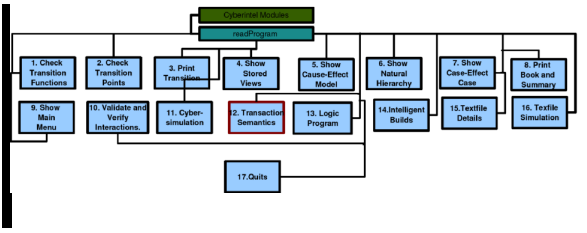
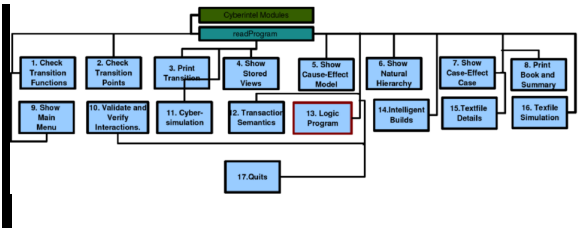
The program should be running by now. Using the keyboard one can interactively execute the program from the input menu-items as shown at display.

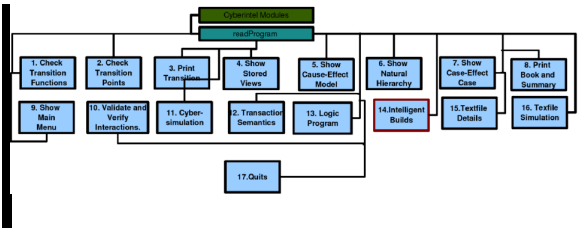
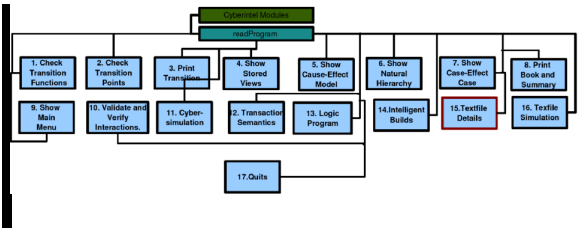
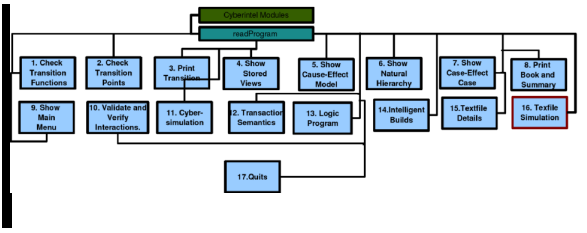
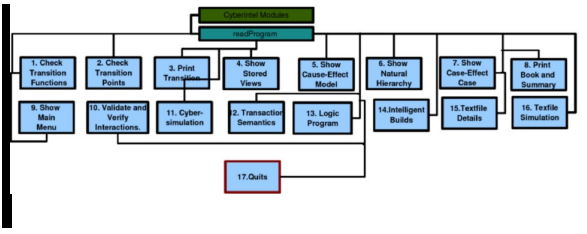
Execution State Number	***Active Functional Module***
1	

Cyberintel Programs Computer Programs

Execution State Number	Active Functional Module
2	
3	
4	
5	

Execution State Number	Active Functional Module
6	
7	
8	
9	

Execution State Number	Active Functional Module
10	
11	
12	
13	

Execution State Number	Active Functional Module
14	
15	
16	
17	

2.2.2 Program Debugging

In debugging cyberintel, a *probe* made of a logging code is inserted after every *print* codeline through out the cyberintelldev script code to create cyberintelldebug script code for debugging purposes. The logging with *debug level* is outputed into a file external to the execution thread. Every execution line of code with *print* instructions has an inserted *logging.debug* instructions. A probe is a statement inserted into a program.

2.3 Summary

In summary, this chapter discussed the implementation phase of cyberintel program codes. The program code and their pseudocodes of the 17 problem defintions were listed in descrbing the computer programs of cyberintel.

BIBLIOGRAPHY

[1]Guido Van Rossum, Python, www.python.org (Accessed 2019).
[2]Introduction to Pascal, Art and Science of Programming.
[3]James F. Peters and Witold Pedrycz(2000). Software Engineering, An Engineering Approach. John Wiley and Sons, USA.
[4] Appiah F. (2019). Cybernetics Intelligence, Lulu Publishing, http://www.lulu.com, USA.
[5]MySQL, http://www.mysql.com (Accessed 2019).
[6]Appiah F.(2019), Cyberintel with Python, Lulu Publishing.

APPENDIX A

CVS DATA.

PYTHON VERSION COMMANDLINE.

PYTHON HELP.

PYTHON MODULE USAGE IN CYBERINTEL.

PYTHON VERSION COMMANDLINE.

```
$ python -V
```

Python 2.5

PYTHON HELP.

$ python --help

```
usage: python [option] ... [-c cmd | -m mod | file |
-] [arg] ...
Options and arguments (and corresponding environment
variables):
-c cmd : program passed in as string (terminates
option list)
-d     : debug output from parser (also
PYTHONDEBUG=x)
-E     : ignore environment variables (such as
PYTHONPATH)
-h     : print this help message and exit (also --
help)
-i     : inspect interactively after running script,
(also PYTHONINSPECT=x)
         and force prompts, even if stdin does not
appear to be a terminal
-m mod : run library module as a script (terminates
option list)
-O     : optimize generated bytecode (a tad; also
PYTHONOPTIMIZE=x)
-OO    : remove doc-strings in addition to the -O
optimizations
-Q arg : division options: -Qold (default), -Qwarn,
-Qwarnall, -Qnew
```

```
-S     : don't imply 'import site' on initialization
-t     : issue warnings about inconsistent tab usage
(-tt: issue errors)
-u     : unbuffered binary stdout and stderr (also
PYTHONUNBUFFERED=x)
         see man page for details on internal
buffering relating to '-u'
-v     : verbose (trace import statements) (also
PYTHONVERBOSE=x)
-V     : print the Python version number and exit
(also --version)
-W arg : warning control (arg is
action:message:category:module:lineno)
-x     : skip first line of source, allowing use of
non-Unix forms of #!cmd
file   : program read from script file
-      : program read from stdin (default;
interactive mode if a tty)
arg ...: arguments passed to program in sys.argv[1:]
Other environment variables:
PYTHONSTARTUP: file executed on interactive startup
(no default)
PYTHONPATH   : ':'-separated list of directories
prefixed to the
               default module search path.  The
result is sys.path.
PYTHONHOME   : alternate <prefix> directory (or
<prefix>:<exec_prefix>).
```

The default module search path uses
<prefix>/pythonX.X.
PYTHONCASEOK : ignore case in 'import' statements
(Windows).

PYTHON MODULE USAGE IN CYBERINTEL.

[cyberintell]$ python -xv cyberintelldev.py

```
# installing zipimport hook
import zipimport # builtin
# installed zipimport hook

# /usr/lib/python2.5/site.pyc matches
/usr/lib/python2.5/site.py
import site # precompiled from
/usr/lib/python2.5/site.pyc

# /usr/lib/python2.5/os.pyc matches
/usr/lib/python2.5/os.py
import os # precompiled from
/usr/lib/python2.5/os.pyc

import posix # builtin
# /usr/lib/python2.5/posixpath.pyc matches
/usr/lib/python2.5/posixpath.py
import posixpath # precompiled from
/usr/lib/python2.5/posixpath.pyc

# /usr/lib/python2.5/stat.pyc matches
/usr/lib/python2.5/stat.py
import stat # precompiled from
/usr/lib/python2.5/stat.pyc

# /usr/lib/python2.5/UserDict.pyc matches
/usr/lib/python2.5/UserDict.py
import UserDict # precompiled from
/usr/lib/python2.5/UserDict.pyc

# /usr/lib/python2.5/copy_reg.pyc matches
/usr/lib/python2.5/copy_reg.py
import copy_reg # precompiled from
/usr/lib/python2.5/copy_reg.pyc
```

```
# /usr/lib/python2.5/types.pyc matches
/usr/lib/python2.5/types.py
import types # precompiled from
/usr/lib/python2.5/types.pyc
import _types # builtin

# /usr/lib/python2.5/warnings.pyc matches
/usr/lib/python2.5/warnings.py
import warnings # precompiled from
/usr/lib/python2.5/warnings.pyc

# /usr/lib/python2.5/linecache.pyc matches
/usr/lib/python2.5/linecache.py
import linecache # precompiled from
/usr/lib/python2.5/linecache.pyc

import encodings # directory
/usr/lib/python2.5/encodings
# /usr/lib/python2.5/encodings/__init__.pyc matches
/usr/lib/python2.5/encodings/__init__.py
import encodings # precompiled from
/usr/lib/python2.5/encodings/__init__.pyc

# /usr/lib/python2.5/codecs.pyc matches
/usr/lib/python2.5/codecs.py
import codecs # precompiled from
/usr/lib/python2.5/codecs.pyc
import _codecs # builtin

# /usr/lib/python2.5/encodings/aliases.pyc matches
/usr/lib/python2.5/encodings/aliases.py
import encodings.aliases # precompiled from
/usr/lib/python2.5/encodings/aliases.pyc

# /usr/lib/python2.5/encodings/utf_8.pyc matches
/usr/lib/python2.5/encodings/utf_8.py
import encodings.utf_8 # precompiled from
/usr/lib/python2.5/encodings/utf_8.pyc
Python 2.5 (r25:51908, Apr 10 2007, 10:29:13)
[GCC 4.1.2 20070403 (Red Hat 4.1.2-8)] on linux2
Type "help", "copyright", "credits" or "license" for
more information.
```

```
cyberintelldev.py: inconsistent use of tabs and
spaces in indentation

import sqlite3 # directory
/usr/lib/python2.5/sqlite3
# /usr/lib/python2.5/sqlite3/__init__.pyc matches
/usr/lib/python2.5/sqlite3/__init__.py
import sqlite3 # precompiled from
/usr/lib/python2.5/sqlite3/__init__.pyc
# /usr/lib/python2.5/sqlite3/dbapi2.pyc matches
/usr/lib/python2.5/sqlite3/dbapi2.py
import sqlite3.dbapi2 # precompiled from
/usr/lib/python2.5/sqlite3/dbapi2.pyc

dlopen("/usr/lib/python2.5/lib-dynload/datetime.so",
2);
import datetime # dynamically loaded from
/usr/lib/python2.5/lib-dynload/datetime.so
dlopen("/usr/lib/python2.5/lib-
dynload/timemodule.so", 2);
import time # dynamically loaded from
/usr/lib/python2.5/lib-dynload/timemodule.so

dlopen("/usr/lib/python2.5/lib-dynload/_sqlite3.so",
2);
import _sqlite3 # dynamically loaded from
/usr/lib/python2.5/lib-dynload/_sqlite3.so

# /usr/lib/python2.5/unittest.pyc matches
/usr/lib/python2.5/unittest.py
import unittest # precompiled from
/usr/lib/python2.5/unittest.pyc

# /usr/lib/python2.5/traceback.pyc matches
/usr/lib/python2.5/traceback.py
import traceback # precompiled from
/usr/lib/python2.5/traceback.pyc

# /usr/lib/python2.5/profile.pyc matches
/usr/lib/python2.5/profile.py
import profile # precompiled from
/usr/lib/python2.5/profile.pyc
import marshal # builtin
```

```
# /usr/lib/python2.5/optparse.pyc matches
/usr/lib/python2.5/optparse.py
import optparse # precompiled from
/usr/lib/python2.5/optparse.pyc

# /usr/lib/python2.5/textwrap.pyc matches
/usr/lib/python2.5/textwrap.py
import textwrap # precompiled from
/usr/lib/python2.5/textwrap.pyc

# /usr/lib/python2.5/string.pyc matches
/usr/lib/python2.5/string.py
import string # precompiled from
/usr/lib/python2.5/string.pyc

# /usr/lib/python2.5/re.pyc matches
/usr/lib/python2.5/re.py
import re # precompiled from
/usr/lib/python2.5/re.pyc

# /usr/lib/python2.5/sre_compile.pyc matches
/usr/lib/python2.5/sre_compile.py
import sre_compile # precompiled from
/usr/lib/python2.5/sre_compile.pyc
import _sre # builtin

# /usr/lib/python2.5/sre_constants.pyc matches
/usr/lib/python2.5/sre_constants.py
import sre_constants # precompiled from
/usr/lib/python2.5/sre_constants.pyc

# /usr/lib/python2.5/sre_parse.pyc matches
/usr/lib/python2.5/sre_parse.py
import sre_parse # precompiled from
/usr/lib/python2.5/sre_parse.pyc

dlopen("/usr/lib/python2.5/lib-
dynload/stropmodule.so", 2);
import strop # dynamically loaded from
/usr/lib/python2.5/lib-dynload/stropmodule.so

# /usr/lib/python2.5/gettext.pyc matches
```

```
/usr/lib/python2.5/gettext.py
import gettext # precompiled from
/usr/lib/python2.5/gettext.pyc

# /usr/lib/python2.5/locale.pyc matches
/usr/lib/python2.5/locale.py
import locale # precompiled from
/usr/lib/python2.5/locale.pyc
dlopen("/usr/lib/python2.5/lib-
dynload/_localemodule.so", 2);
import _locale # dynamically loaded from
/usr/lib/python2.5/lib-dynload/_localemodule.so

dlopen("/usr/lib/python2.5/lib-dynload/operator.so",
2);
import operator # dynamically loaded from
/usr/lib/python2.5/lib-dynload/operator.so
# /usr/lib/python2.5/copy.pyc matches
/usr/lib/python2.5/copy.py
import copy # precompiled from
/usr/lib/python2.5/copy.pyc

# /usr/lib/python2.5/struct.pyc matches
/usr/lib/python2.5/struct.py
import struct # precompiled from
/usr/lib/python2.5/struct.pyc
dlopen("/usr/lib/python2.5/lib-dynload/_struct.so",
2);
import _struct # dynamically loaded from
/usr/lib/python2.5/lib-dynload/_struct.so
import errno # builtin

dlopen("/usr/lib/python2.5/lib-dynload/resource.so",
2);
import resource # dynamically loaded from
/usr/lib/python2.5/lib-dynload/resource.so

dlopen("/usr/lib/python2.5/site-packages/_mysql.so",
2);
# /usr/lib/python2.5/site-
packages/_mysql_exceptions.pyc matches
/usr/lib/python2.5/site-
packages/_mysql_exceptions.py
```

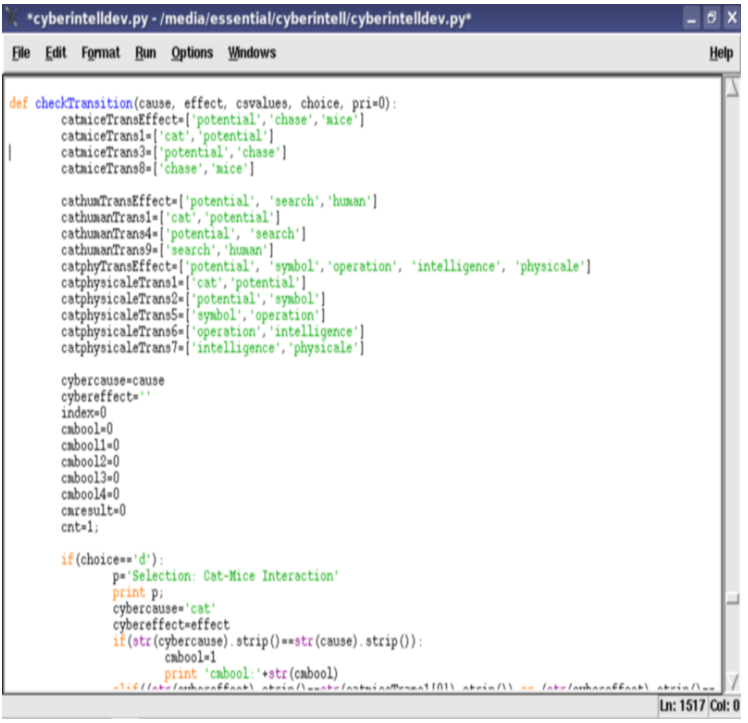
```
import _mysql_exceptions # precompiled from
/usr/lib/python2.5/site-
packages/_mysql_exceptions.pyc
import _mysql # dynamically loaded from
/usr/lib/python2.5/site-packages/_mysql.so

# /usr/lib/python2.5/pprint.pyc matches
/usr/lib/python2.5/pprint.py
import pprint # precompiled from
/usr/lib/python2.5/pprint.pyc

dlopen("/usr/lib/python2.5/lib-
dynload/cStringIO.so", 2);
import cStringIO # dynamically loaded from
/usr/lib/python2.5/lib-dynload/cStringIO.so
```

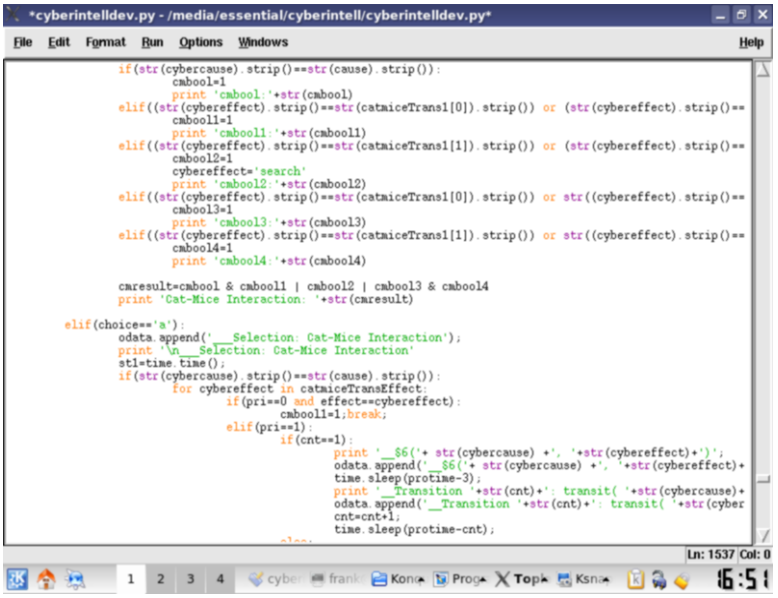
APPENDIX B

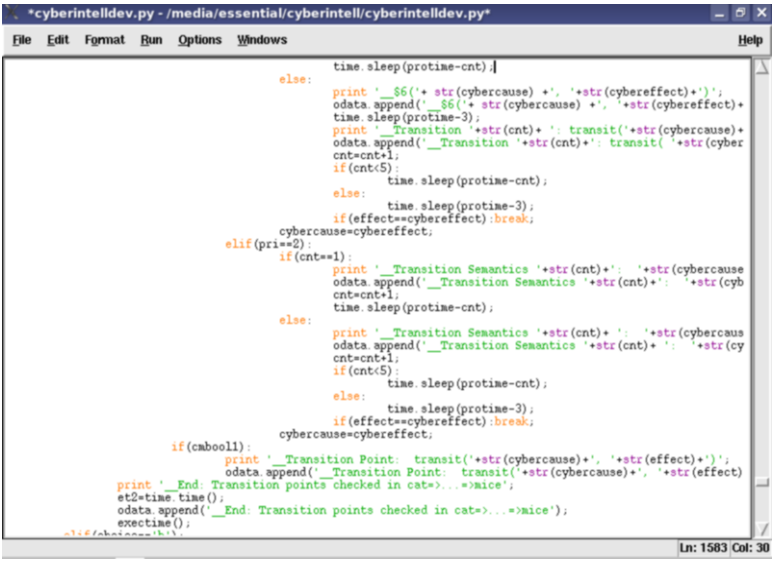
Idle Pseudocodes:

These are the pseudocode extras from the program codes above in Chapter 2.

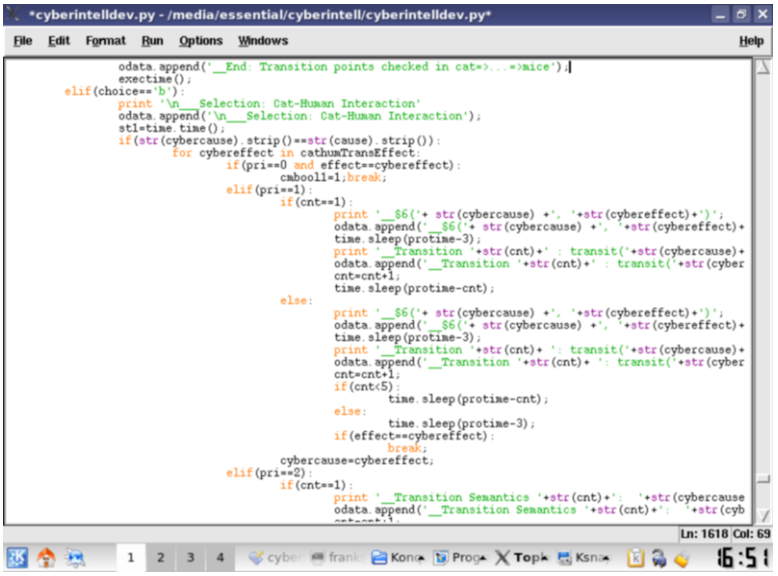
`checktransition Python Shell`

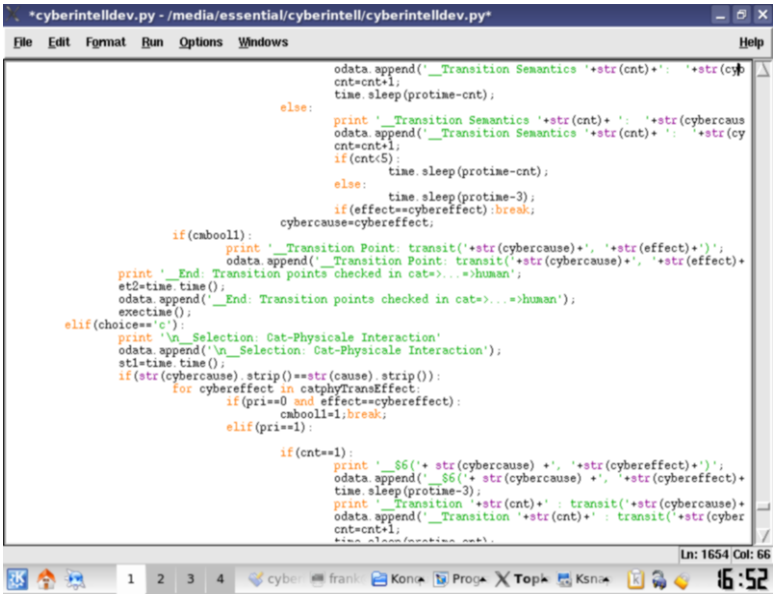


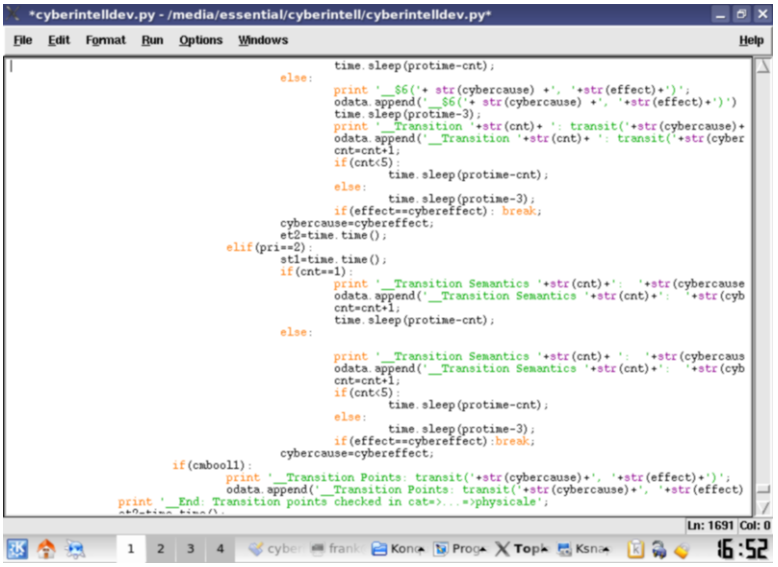


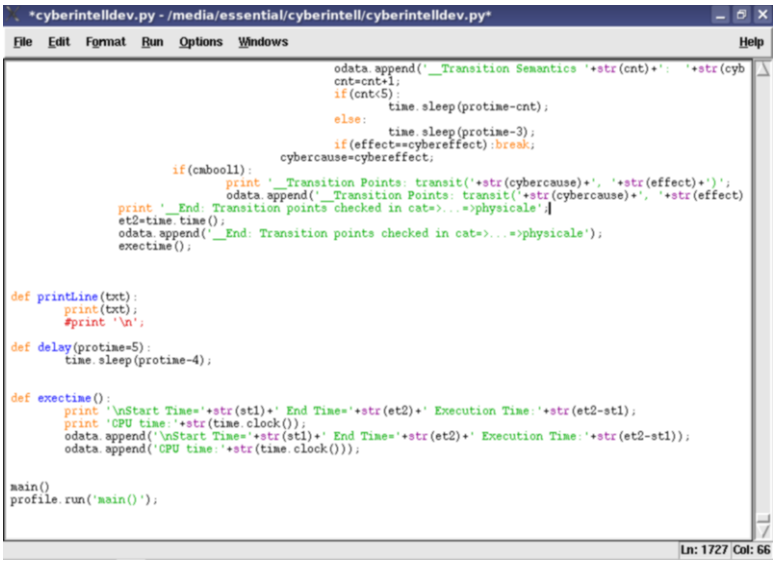


Cyberintel Programs Computer Programs

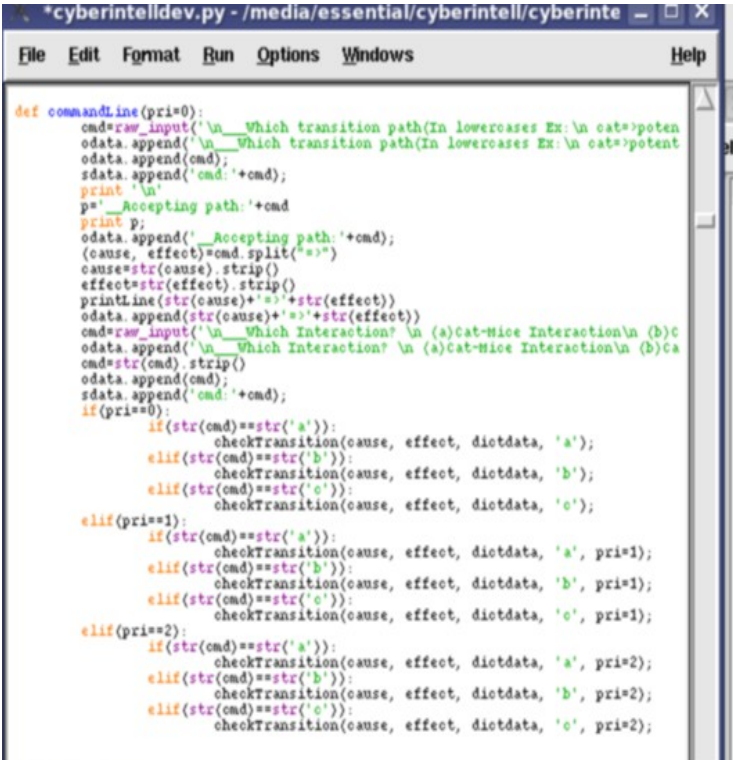

cyberintelldev.py - /media/essential/cyberintell/cyberintelldev.py
File Edit Format Run Options Windows Help
odata.append('__End: Transition points checked in cat=>...=>mice');
exectime();
elif(choice=='b'):
print '\n__Selection: Cat-Human Interaction'
odata.append('\n__Selection: Cat-Human Interaction');
stl=time.time();
if(str(cybercause).strip()==str(cause).strip()):
for cybereffect in cathumTransEffect:
if(pri==0 and effect==cybereffect):
cmbool1=1;break;
elif(pri==1):
if(cnt==1):
print '__$6('+ str(cybercause) +', '+str(cybereffect)+')';
odata.append('__$6('+ str(cybercause) +', '+str(cybereffect)+
time.sleep(protime-3);
print '__Transition '+str(cnt)+' : transit('+str(cybercause)+
odata.append('__Transition '+str(cnt)+' : transit('+str(cyber
cnt=cnt+1;
time.sleep(protime-cnt);
else:
print '__$6('+ str(cybercause) +', '+str(cybereffect)+')';
odata.append('__$6('+ str(cybercause) +', '+str(cybereffect)+
time.sleep(protime-3);
print '__Transition '+str(cnt)+ ': transit('+str(cybercause)+
odata.append('__Transition '+str(cnt)+ ': transit('+str(cyber
cnt=cnt+1;
if(cnt<5):
time.sleep(protime-cnt);
else:
time.sleep(protime-3);
if(effect==cybereffect):
break;
cybercause=cybereffect;
elif(pri==2):
if(cnt==1):
print '__Transition Semantics '+str(cnt)+': '+str(cybercause
odata.append('__Transition Semantics '+str(cnt)+': '+str(cyb
Ln: 1618 Col: 69
1 2 3 4
16:51


cyberintelldev.py - /media/essential/cyberintell/cyberintelldev.py
File Edit Format Run Options Windows Help
odata.append('__Transition Semantics '+str(cnt)+': '+str(cyb
cnt=cnt+1;
time.sleep(protime-cnt);
else:
print '__Transition Semantics '+str(cnt)+ ': '+str(cybercaus
odata.append('__Transition Semantics '+str(cnt)+ ': '+str(cy
cnt=cnt+1;
if(cnt<5):
time.sleep(protime-cnt);
else:
time.sleep(protime-3);
if(effect==cybereffect):break;
cybercause=cybereffect;
if(cmbool1):
print '__Transition Point: transit('+str(cybercause)+', '+str(effect)+')';
odata.append('__Transition Point: transit('+str(cybercause)+', '+str(effect)+
print '__End: Transition points checked in cat=>...=>human';
et2=time.time();
odata.append('__End: Transition points checked in cat=>...=>human');
exectime();
elif(choice=='c'):
print '\n__Selection: Cat-Physicale Interaction'
odata.append('\n__Selection: Cat-Physicale Interaction');
st1=time.time();
if(str(cybercause).strip()==str(cause).strip()):
for cybereffect in catphyTransEffect:
if(pri==0 and effect==cybereffect):
cmbool1=1;break;
elif(pri==1):
if(cnt==1):
print '__$6('+ str(cybercause) +', '+str(cybereffect)+')';
odata.append('__$6('+ str(cybercause) +', '+str(cybereffect)+
time.sleep(protime-3);
print '__Transition '+str(cnt)+' : transit('+str(cybercause)+
odata.append('__Transition '+str(cnt)+' : transit('+str(cyber
cnt=cnt+1;
Ln: 1654 Col: 66
1 2 3 4
16:52


cyberintelldev.py - /media/essential/cyberintell/cyberintelldev.py
File Edit Format Run Options Windows Help
time.sleep(protime-cnt);
else:
print '__$6('+ str(cybercause) +', '+str(effect)+')';
odata.append('__$6('+ str(cybercause) +', '+str(effect)+')')
time.sleep(protime-3);
print '__Transition '+str(cnt)+ ': transit('+str(cybercause)+
odata.append('__Transition '+str(cnt)+ ': transit('+str(cyber
cnt=cnt+1;
if(cnt<5):
time.sleep(protime-cnt);
else:
time.sleep(protime-3);
if(effect==cybereffect): break;
cybercause=cybereffect;
et2=time.time();
elif(pri==2):
st1=time.time();
if(cnt==1):
print '__Transition Semantics '+str(cnt)+': '+str(cybercause
odata.append('__Transition Semantics '+str(cnt)+': '+str(cyb
cnt=cnt+1;
time.sleep(protime-cnt);
else:
print '__Transition Semantics '+str(cnt)+ ': '+str(cybercaus
odata.append('__Transition Semantics '+str(cnt)+': '+str(cyb
cnt=cnt+1;
if(cnt<5):
time.sleep(protime-cnt);
else:
time.sleep(protime-3);
if(effect==cybereffect):break;
cybercause=cybereffect;
if(cmbool1):
print '__Transition Points: transit('+str(cybercause)+', '+str(effect)+')';
odata.append('__Transition Points: transit('+str(cybercause)+', '+str(effect)
print '__End: Transition points checked in cat=>...=>physicale';
Ln: 1691 Col: 0
1 2 3 4
16:52

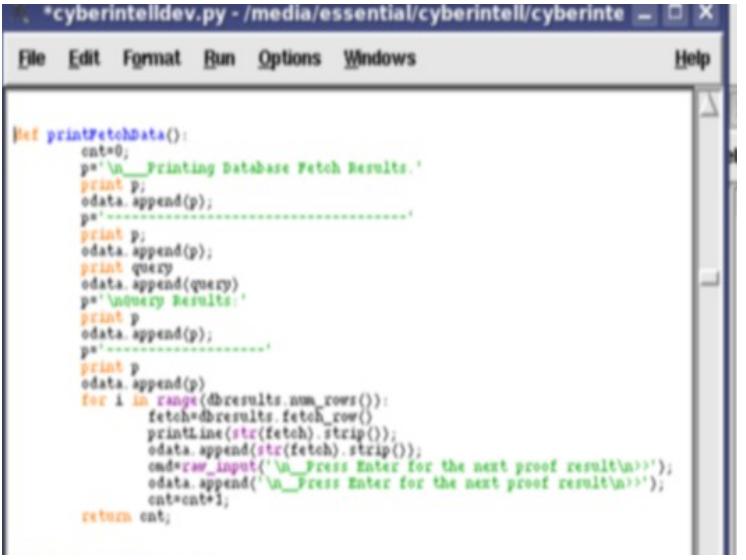

cyberintelldev.py - /media/essential/cyberintell/cyberintelldev.py
File Edit Format Run Options Windows Help
odata.append('__Transition Semantics '+str(cnt)+': '+str(cyb
cnt=cnt+1;
if(cnt<5):
time.sleep(protime-cnt);
else:
time.sleep(protime-3);
if(effect==cybereffect):break;
cybercause=cybereffect;
if(cmbool1):
print '__Transition Points: transit('+str(cybercause)+', '+str(effect)+')';
odata.append('__Transition Points: transit('+str(cybercause)+', '+str(effect)
print '__End: Transition points checked in cat=>...=>physicale';
et2=time.time();
odata.append('__End: Transition points checked in cat=>...=>physicale');
exectime();
def printLine(txt):
print(txt);
#print '\n';
def delay(protime=5):
time.sleep(protime-4);
def exectime():
print '\nStart Time='+str(st1)+' End Time='+str(et2)+' Execution Time:'+str(et2-st1);
print 'CPU time:'+str(time.clock());
odata.append('\nStart Time='+str(st1)+' End Time='+str(et2)+' Execution Time:'+str(et2-st1));
odata.append('CPU time:'+str(time.clock()));
main()
profile.run('main()');
Ln: 1727 Col: 66

commandline Python Shell

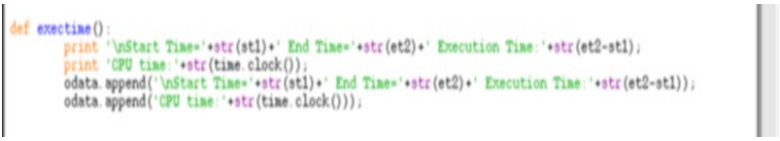


querydatabase Python Shell

printfetchdata Python Shell

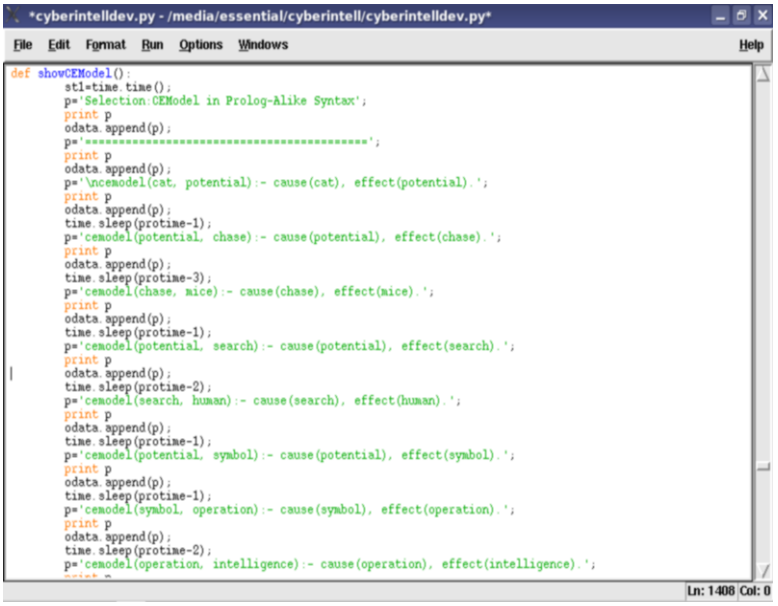


execTime Python Shell



```
def exectime():
        print '\nStart Time='+str(st1)+' End Time='+str(et2)+' Execution Time:'+str(et2-st1);
        print 'CPU time:'+str(time.clock());
        odata.append('\nStart Time='+str(st1)+' End Time='+str(et2)+' Execution Time:'+str(et2-st1));
        odata.append('CPU time:'+str(time.clock()));
```

showCEModel Python Shell



cyberintelldev.py - /media/essential/cyberintell/cyberintelldev.py

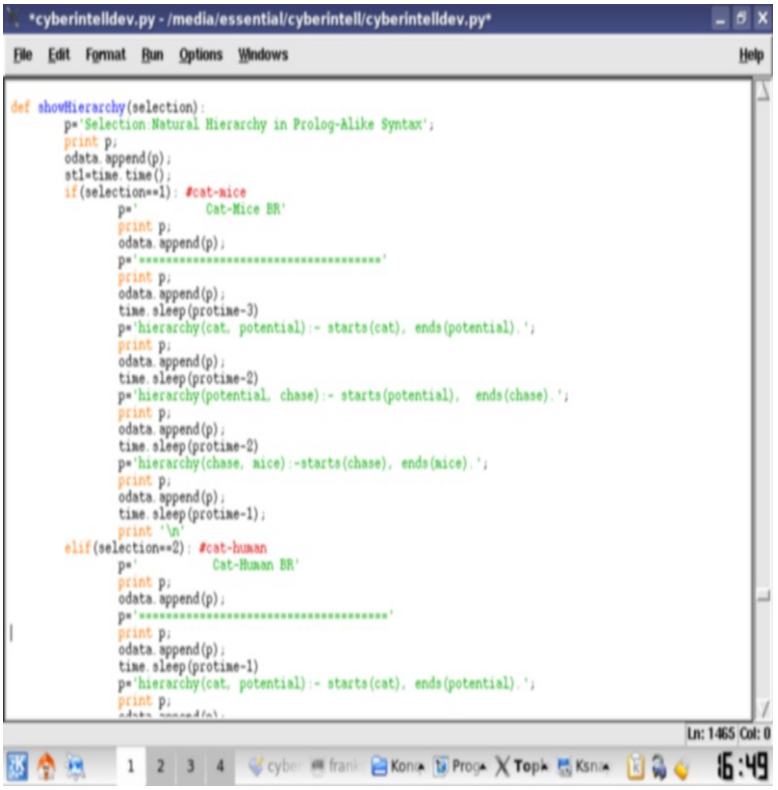
File Edit Format Run Options Windows Help

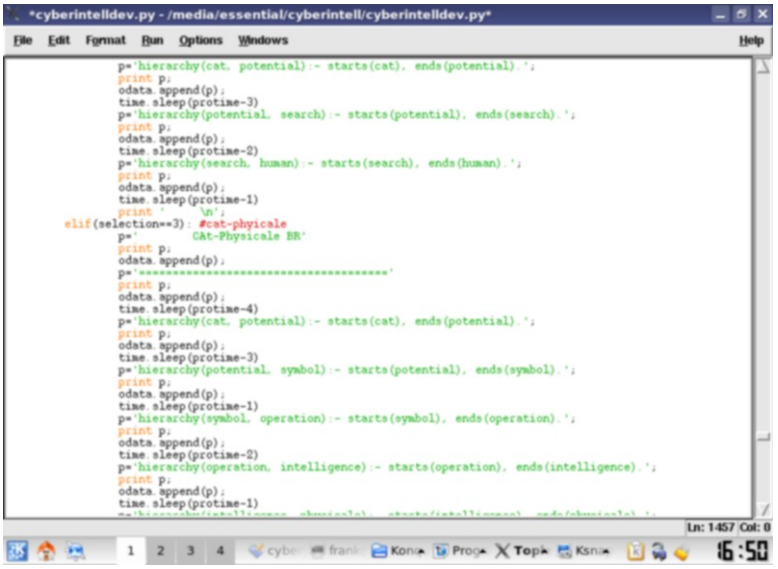
```
def showCEModel():
        st1=time.time();
        p='Selection:CEModel in Prolog-Alike Syntax';
        print p
        odata.append(p);
        p='========================================';
        print p
        odata.append(p);
        p='\ncemodel(cat, potential):- cause(cat), effect(potential).';
        print p
        odata.append(p);
        time.sleep(protime-1);
        p='cemodel(potential, chase):- cause(potential), effect(chase).';
        print p
        odata.append(p);
        time.sleep(protime-3);
        p='cemodel(chase, mice):- cause(chase), effect(mice).';
        print p
        odata.append(p);
        time.sleep(protime-1);
        p='cemodel(potential, search):- cause(potential), effect(search).';
        print p
        odata.append(p);
        time.sleep(protime-2);
        p='cemodel(search, human):- cause(search), effect(human).';
        print p
        odata.append(p);
        time.sleep(protime-1);
        p='cemodel(potential, symbol):- cause(potential), effect(symbol).';
        print p
        odata.append(p);
        time.sleep(protime-1);
        p='cemodel(symbol, operation):- cause(symbol), effect(operation).';
        print p
        odata.append(p);
        time.sleep(protime-2);
        p='cemodel(operation, intelligence):- cause(operation), effect(intelligence).';
```

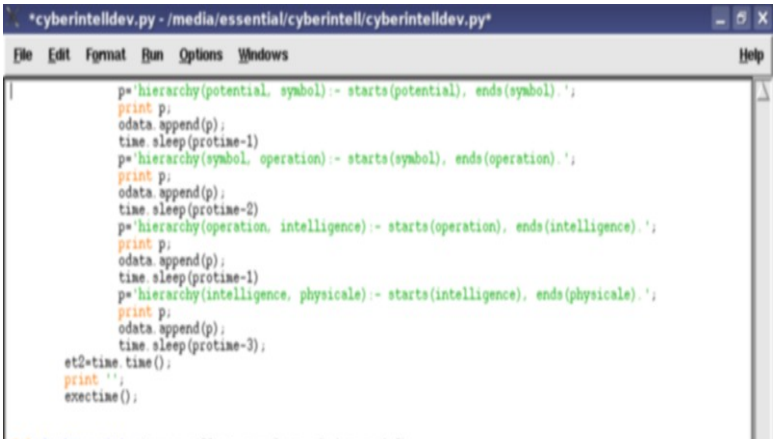
Ln: 1408 Col: 0

```
        time.sleep(protime-2);
        p='cemodel(operation, intelligence):- cause(operation), effect(intelligence).';
        print p
        odata.append(p);
        time.sleep(protime-3);
        p='cemodel(intelligence, physicale):- cause(intelligence), effect(physicale).';
        print p
        odata.append(p);
        et2=time.time();
        p=''
        exectime();
```

showHierarchy Python Shell

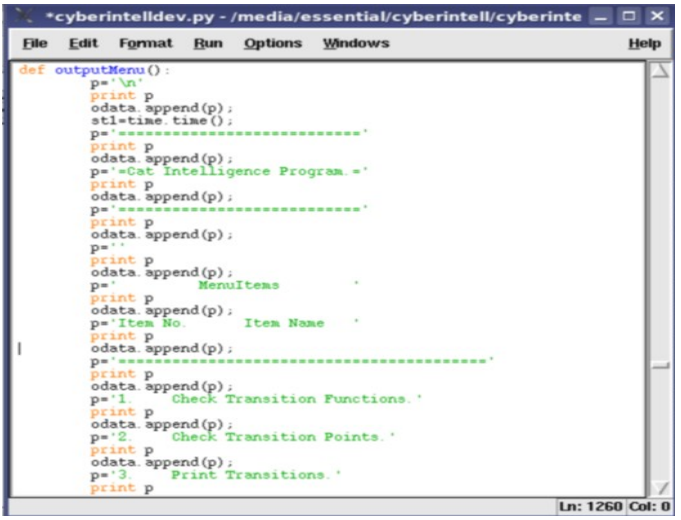


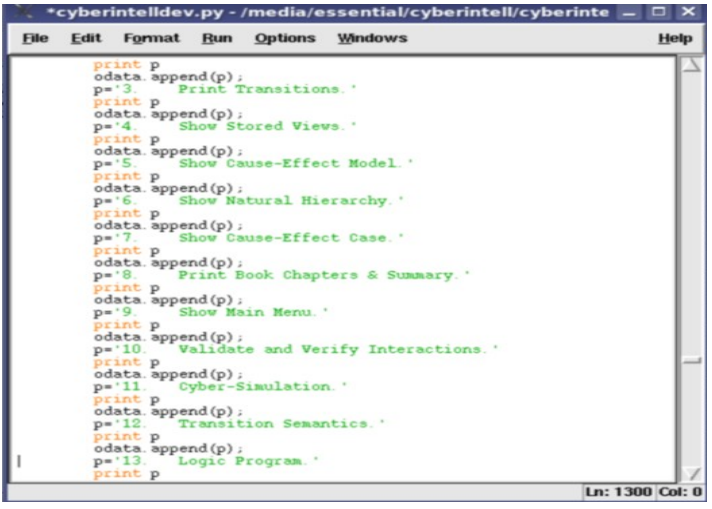

cyberintelldev.py - /media/essential/cyberintell/cyberintelldev.py
File Edit Format Run Options Windows Help
p='hierarchy(cat, potential):- starts(cat), ends(potential).';
print p;
odata.append(p);
time.sleep(protime-3)
p='hierarchy(potential, search):- starts(potential), ends(search).';
print p;
odata.append(p);
time.sleep(protime-2)
p='hierarchy(search, human):- starts(search), ends(human).';
print p;
odata.append(p);
time.sleep(protime-1)
print '\n';
elif(selection==3): #cat-phyicale
p='CAt-Physicale BR'
print p;
odata.append(p);
p='======================================'
print p;
odata.append(p);
time.sleep(protime-4)
p='hierarchy(cat, potential):- starts(cat), ends(potential).';
print p;
odata.append(p);
time.sleep(protime-3)
p='hierarchy(potential, symbol):- starts(potential), ends(symbol).';
print p;
odata.append(p);
time.sleep(protime-1)
p='hierarchy(symbol, operation):- starts(symbol), ends(operation).';
print p;
odata.append(p);
time.sleep(protime-2)
p='hierarchy(operation, intelligence):- starts(operation), ends(intelligence).';
print p;
odata.append(p);
time.sleep(protime-1)
Ln: 1457 Col: 0
1 2 3 4
16:50


cyberintelldev.py - /media/essential/cyberintell/cyberintelldev.py
File Edit Format Run Options Windows Help
p='hierarchy(potential, symbol):- starts(potential), ends(symbol).';
print p;
odata.append(p);
time.sleep(protime-1)
p='hierarchy(symbol, operation):- starts(symbol), ends(operation).';
print p;
odata.append(p);
time.sleep(protime-2)
p='hierarchy(operation, intelligence):- starts(operation), ends(intelligence).';
print p;
odata.append(p);
time.sleep(protime-1)
p='hierarchy(intelligence, physicale):- starts(intelligence), ends(physicale).';
print p;
odata.append(p);
time.sleep(protime-3);
et2=time.time();
print '';
exectime();

showCECase Python Shell

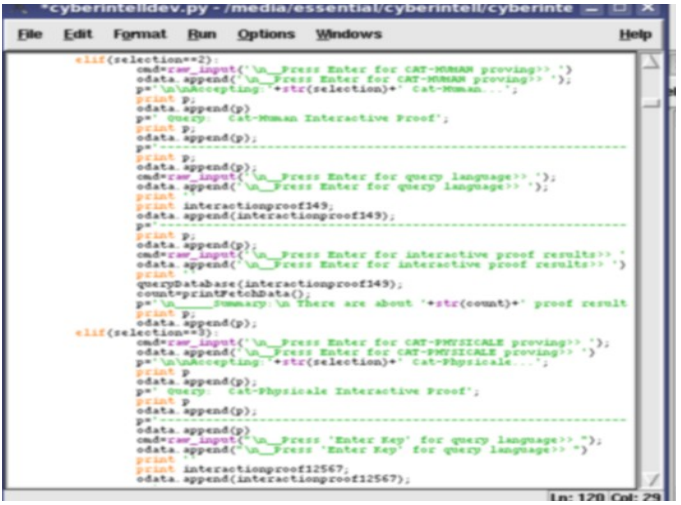
outputMenu Python Shell

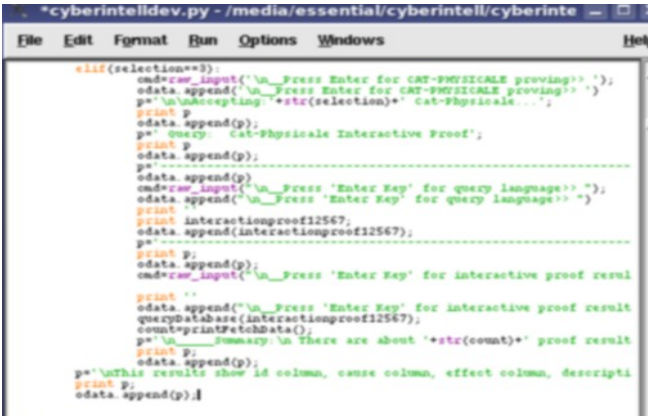




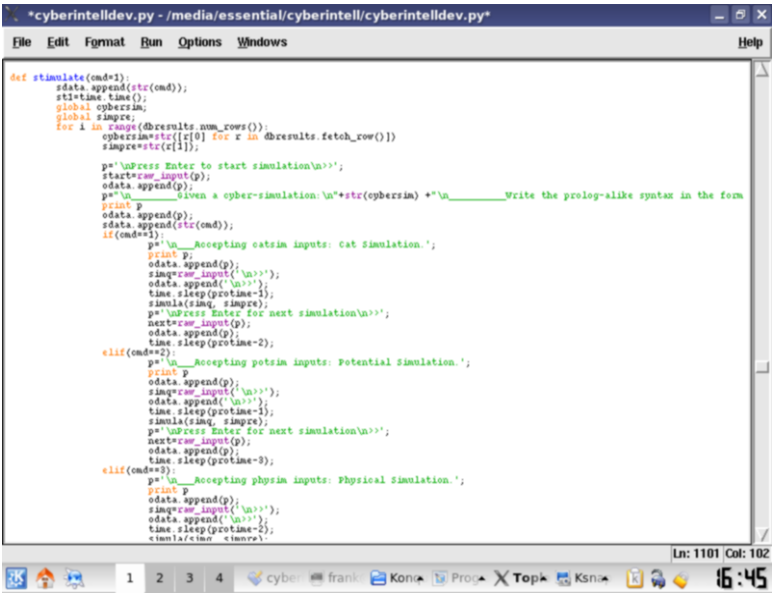
```
print p
odata.append(p);
p='13.    Logic Program.'
print p
odata.append(p);
p='14.    Intelligence Builds.'
print p
odata.append(p);
p='15.    Show Textfile Session.'
print p
odata.append(p);
p='16.    Run TextFile Simulation.'
print p
odata.append(p);
p='17.    Press Esc and Enter to quit.'
print p
odata.append(p);
p='========================================'
et2=time.time();
exectime()
```

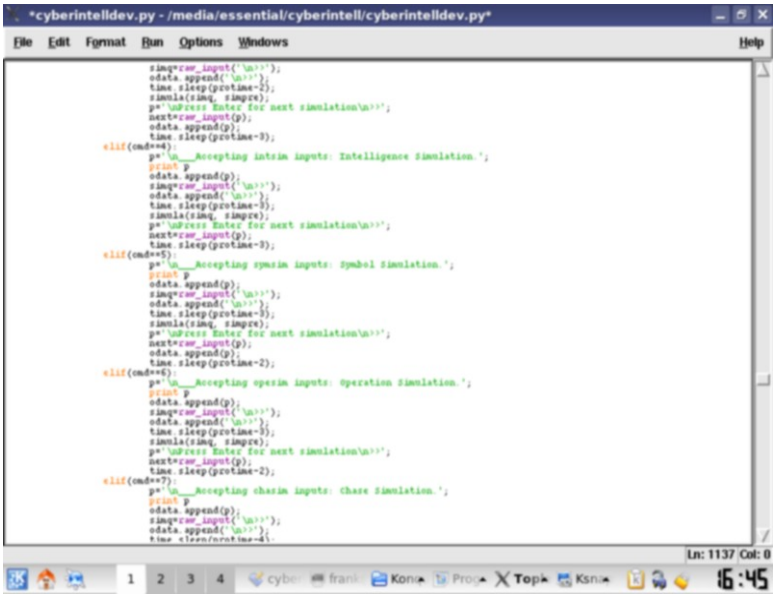
interactiveproof Python Shell

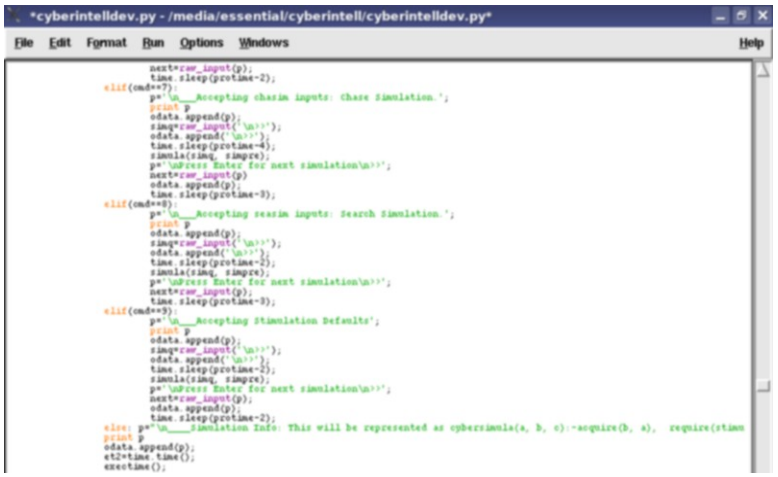

*cyberintelldev.py - /media/essential/cyberintell/cyberinte
File Edit Format Run Options Windows Help
elif(selection==2):
cmd=raw_input('\n__Press Enter for CAT-HUMAN proving>> ')
odata.append('\n__Press Enter for CAT-HUMAN proving>> ');
p='\n\nAccepting: '+str(selection)+' Cat-Human...';
print p;
odata.append(p)
p=' Query: Cat-Human Interactive Proof';
print p;
odata.append(p);
p='----------
print p;
odata.append(p);
cmd=raw_input('\n__Press Enter for query language>> ');
odata.append('\n__Press Enter for query language>> ');
print ''
print interactionproof149;
odata.append(interactionproof149);
p='----------
print p;
odata.append(p);
cmd=raw_input('\n__Press Enter for interactive proof results>> '
odata.append('\n__Press Enter for interactive proof results>> ')
print ''
querydatabase(interactionproof149);
count=printFetchData();
p='\n_____Summary:\n There are about '+str(count)+' proof result
print p;
odata.append(p);
elif(selection==3):
cmd=raw_input('\n__Press Enter for CAT-PHYSICALE proving>> ');
odata.append('\n__Press Enter for CAT-PHYSICALE proving>> ')
p='\n\nAccepting: '+str(selection)+' Cat-Physicale...';
print p
odata.append(p);
p=' Query: Cat-Physicale Interactive Proof';
print p
odata.append(p);
p='----------
odata.append(p)
cmd=raw_input('\n__Press 'Enter Key' for query language>> ');
odata.append('\n__Press 'Enter Key' for query language>> ')
print ''
print interactionproof12567;
odata.append(interactionproof12567);
Ln: 120 Col: 29


*cyberintelldev.py - /media/essential/cyberintell/cyberinte
File Edit Format Run Options Windows
elif(selection==3):
cmd=raw_input('\n__Press Enter for CAT-PHYSICALE proving>> ');
odata.append('\n__Press Enter for CAT-PHYSICALE proving>> ')
p='\n\nAccepting: '+str(selection)+' Cat-Physicale...';
print p
odata.append(p);
p=' Query: Cat-Physicale Interactive Proof';
print p
odata.append(p);
p='----------
odata.append(p)
cmd=raw_input('\n__Press 'Enter Key' for query language>> ');
odata.append('\n__Press 'Enter Key' for query language>> ')
print ''
print interactionproof12567;
odata.append(interactionproof12567);
p='----------
print p;
odata.append(p);
cmd=raw_input('\n__Press 'Enter Key' for interactive proof resul
print ''
odata.append('\n__Press 'Enter Key' for interactive proof result
querydatabase(interactionproof12567);
count=printFetchData();
p='\n_____Summary:\n There are about '+str(count)+' proof result
print p;
odata.append(p);
p='\nThis results show id column, cause column, effect column, descripti
print p;
odata.append(p)

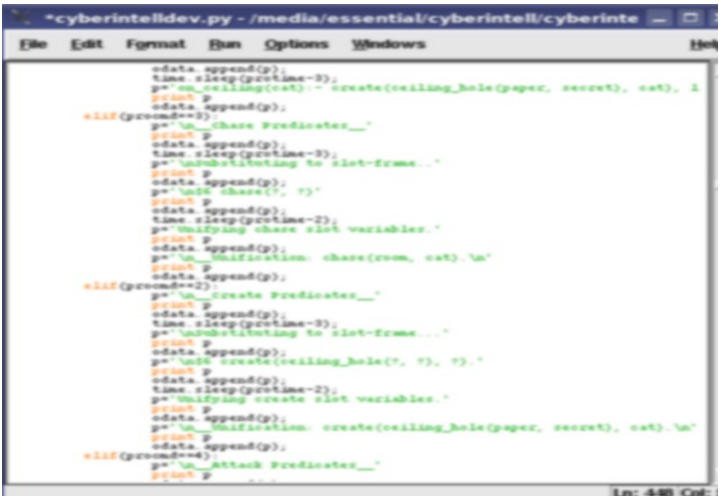
stimulate Python shell




cyberintelldev.py - /media/essential/cyberintell/cyberintelldev.py
File Edit Format Run Options Windows Help
Ln: 1137 Col: 0


cyberintelldev.py - /media/essential/cyberintell/cyberintelldev.py
File Edit Format Run Options Windows Help

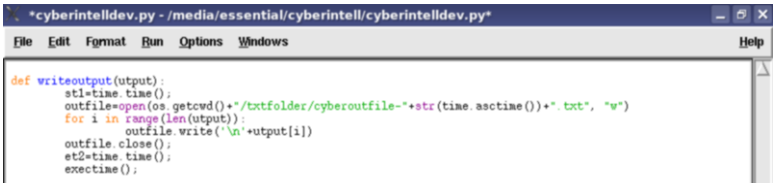
logicprogram Python Shell



*cyberintelldev.py - /media/essential/cyberintell/cyberinte
File Edit Format Run Options Windows
time.sleep(protime-2);
p='attack(human, cat):- chase(human, cat).'
print p
odata.append(p);
time.sleep(protime-2);
p='attack(human, cat):- disentangle(cat, human).'
print p
odata.append(p);
time.sleep(protime-2);
p='attack(human, cat):- disentangle(cat, human), leave(room, cat
print p
odata.append(p);
time.sleep(protime-2);
print p
odata.append(p);
p='attack(human, cat):- chase(human, cat), leave(room, cat).'
time.sleep(protime-2);
p='attack(human, cat):- chase(human, cat), create(ceiling_hole(p
print p
odata.append(p);
elif(proomd==5):
p='\n__Disentangle Predicates__'
print p
odata.append(p);
time.sleep(protime-3);
p='\nSubstituting to slot-frame...'
print p
odata.append(p);
p='\n16 disentangle(?, ?)'
print p
odata.append(p);
time.sleep(protime-2);
p='\nUnifying disentangle slot variables.'
print p
odata.append(p);
p='\n__Unification: disentangle(cat_fight, human).'
print p
odata.append(p);
elif(proomd==6):
p='\n__Escape Predicates__'

*cyberintelldev.py - /media/essential/cyberintell/cyberinte
File Edit Format Run Options Windows
print p
odata.append(p);
elif(proomd==6):
p='\n__Escape Predicates__'
print p
odata.append(p);
time.sleep(protime-3);
p='\nSubstituting to slot-frame...'
print p
odata.append(p);
p='\n16 escape(?, ?)'
print p
odata.append(p);
time.sleep(protime-2);
p='\nUnifying escape slot variables.'
print p
odata.append(p);
p='\n__Unification: escape(hole_net, cat).'
print p
odata.append(p);
elif(proomd==7):
p='\n__Leave Predicates__'
time.sleep(protime-3);
print p
odata.append(p);
p='\nSubstituting to slot-frame...'
print p
odata.append(p);
p='\n16 leave(?, ?)'
print p
odata.append(p);
time.sleep(protime-2);
p='Unifying escape slot variables.'
print p
odata.append(p);
p='\n__Unification: leave(room, cat).'
print p
odata.append(p);
else:
p='Logic Program is not supported!'
print p
odata.append(p);

writeouput Python Shell



writedata Python Shell

Cyberintelldev- Cyberintel AI Programalike (Othercodes)

```
import codes
variable codes
main codes
delay codes
connectdatabase codes
readProgram codes
```

Illustration 4: Import Python Shell

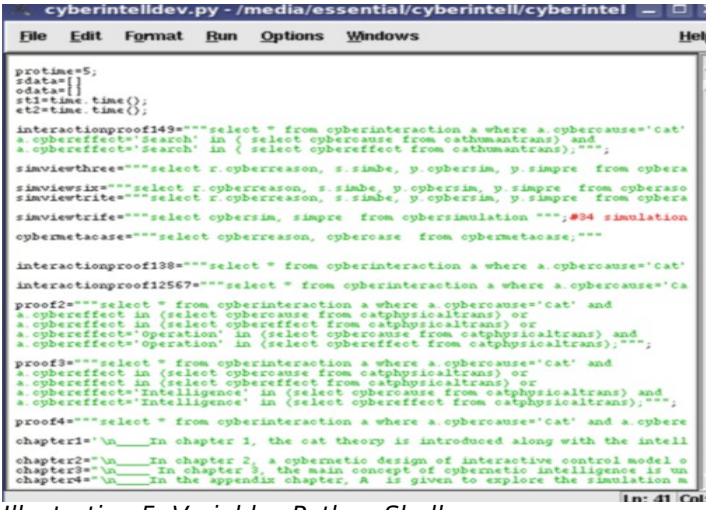

cyberintelldev.py - /media/essential/cyberintell/cyberintel
File Edit Format Run Options Windows
protime=5;
sdata=[]
odata=[]
st1=time.time();
et2=time.time();
interactionproof149="""select * from cyberinteraction a where a.cybercause='Cat'
a.cybereffect='Search' in (select cybercause from cathumantrans) and
a.cybereffect='Search' in (select cybereffect from cathumantrans);""";
simviewthree="""select r.cyberreason, s.simbe, y.cybersim, y.simpre from cybera
simviewsix="""select r.cyberreason, s.simbe, y.cybersim, y.simpre from cyberaso
simviewtrite="""select r.cyberreason, s.simbe, y.cybersim, y.simpre from cybera
simviewtrife="""select cybersim, simpre from cybersimulation """;#84 simulation
cybermetacase="""select cyberreason, cybercase from cybermetacase;"""
interactionproof138="""select * from cyberinteraction a where a.cybercause='Cat'
interactionproof12567="""select * from cyberinteraction a where a.cybercause='Ca
proof2="""select * from cyberinteraction a where a.cybercause='Cat' and
a.cybereffect in (select cybercause from catphysicaltrans) or
a.cybereffect in (select cybereffect from catphysicaltrans) or
a.cybereffect='Operation' in (select cybercause from catphysicaltrans) and
a.cybereffect='Operation' in (select cybereffect from catphysicaltrans);""";
proof3="""select * from cyberinteraction a where a.cybercause='Cat' and
a.cybereffect in (select cybercause from catphysicaltrans) or
a.cybereffect in (select cybereffect from catphysicaltrans) or
a.cybereffect='Intelligence' in (select cybercause from catphysicaltrans) and
a.cybereffect='Intelligence' in (select cybereffect from catphysicaltrans);""";
proof4="""select * from cyberinteraction a where a.cybercause='Cat' and a.cybere
chapter1='\n____In chapter 1, the cat theory is introduced along with the intell
chapter2="\n____In chapter 2, a cybernetic design of interactive control model o
chapter3="\n____In chapter 3, the main concept of cybernetic intelligence is un
chapter4="\n____In the appendix chapter, A is given to explore the simulation m

Illustration 5: Variables Python Shell

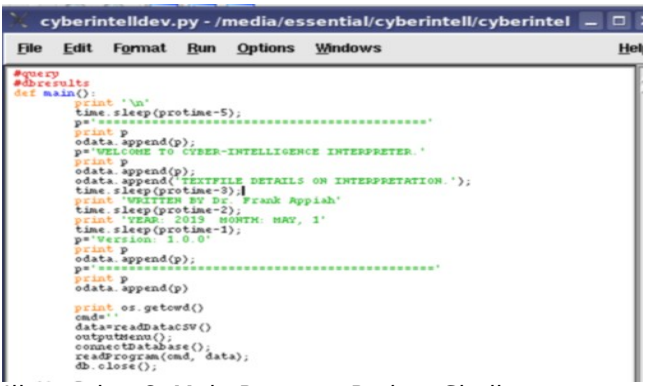


Illustration 6: Main Program Python Shell

```
def delay(protime=5):
        time.sleep(protime-4);
```

Illustration 7: Delay Python Code

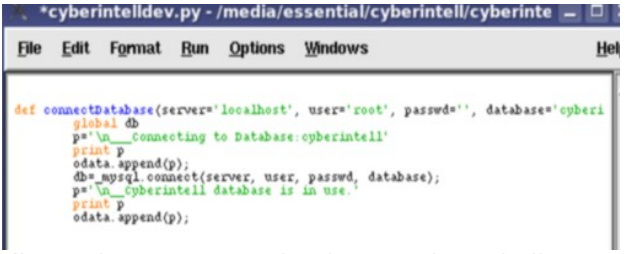


Illustration 8: connectdatabase Python Shell

Illustration 9: readProgram Definition

Illustration 10: readProgram Continue 1

Illustration 11: readProgram Continue 2

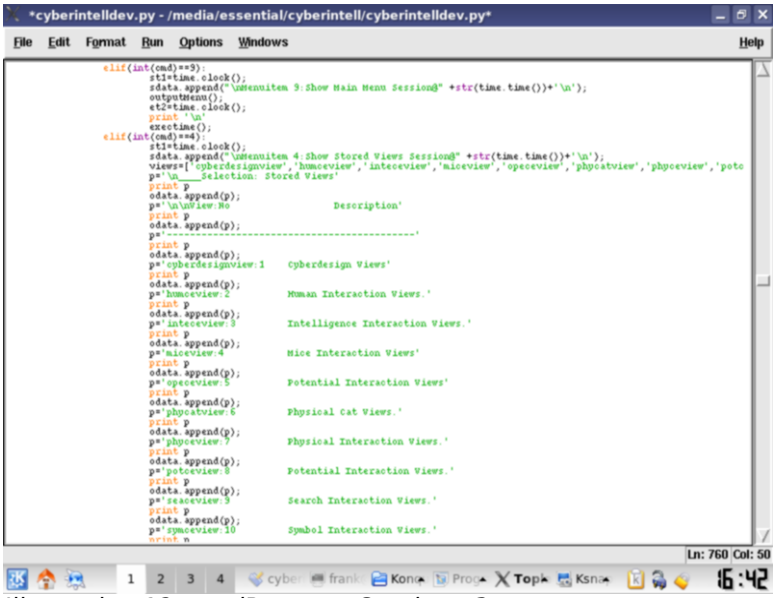


Illustration 12: readProgram Continue 3

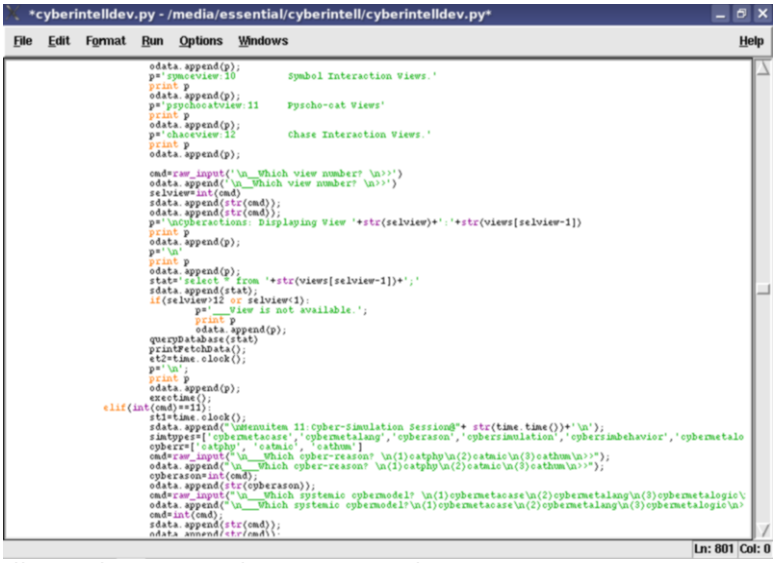

cyberintelldev.py - /media/essential/cyberintell/cyberintelldev.py
File Edit Format Run Options Windows Help
Ln: 801 Col: 0

Illustration 13: readProgram Continue 4

cyberintelldev.py - /media/essential/cyberintell/cyberintelldev.py
File Edit Format Run Options Windows Help
Ln: 846 Col: 0

Illustration 14: readProgram Continue 5

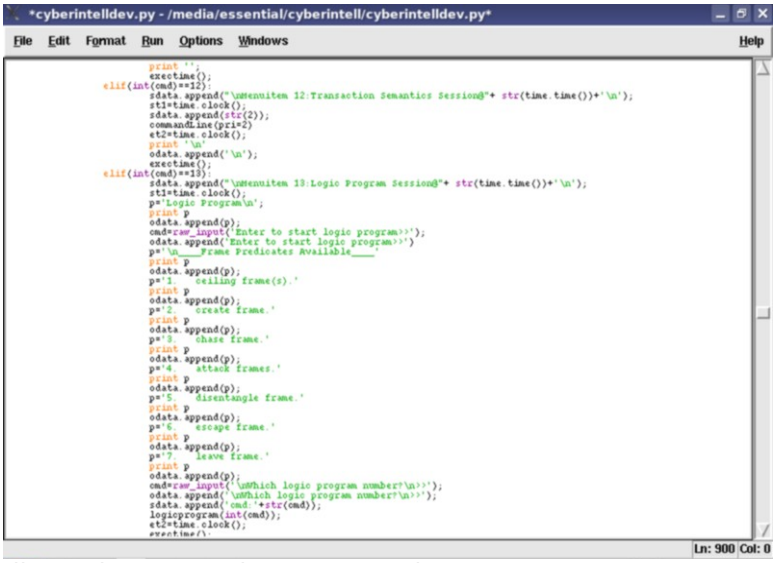


Illustration 15: readProgram Continue 6

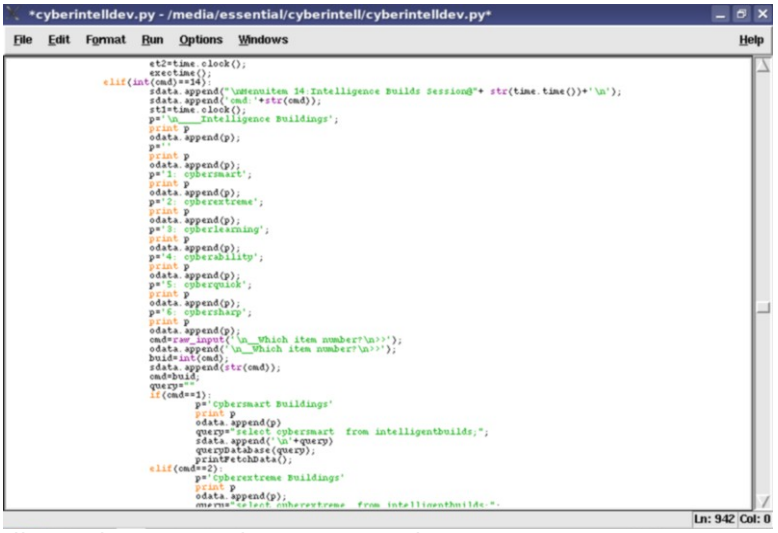


Illustration 16: readProgram Continue 7

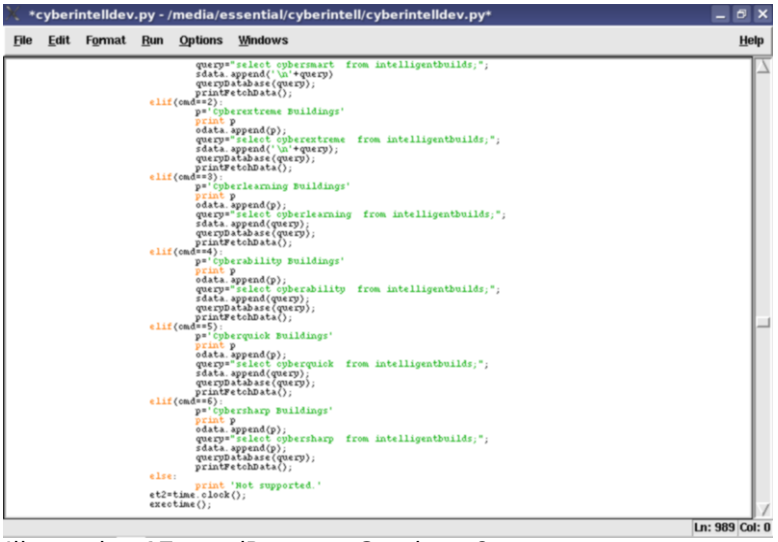


Illustration 17: readProgram Continue 8

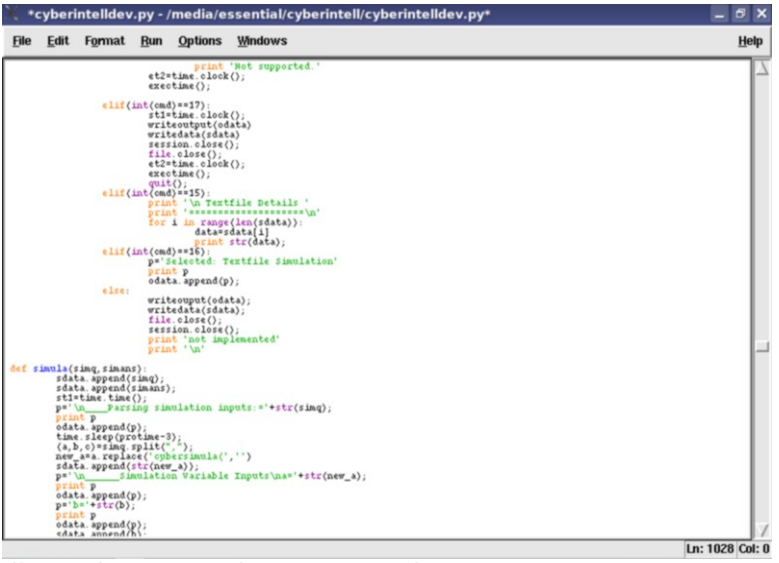


Illustration 18: readProgram Continue 9

Index

book.....3
Book.....19, 33, 56, 96
BOOK.....3
branches.....19, 31, 55, 94
Case.....19, 32, 56, 95
cat.....19, 30, 54, 93
cat-mice.....19, 30, 54, 93
cause. .18, 19, 25, 27, 29, 31, 52, 53, 54, 55, 81, 83, 87, 92, 94
Cause.....19, 29, 32, 54, 56, 92, 95
cause-effect.....19, 31, 55, 94
Cause-Effect.....19, 29, 32, 54, 56, 92, 95
Check.....18, 25, 27, 53, 83
concepts.....19, 28, 54, 88
CVS DATA.....128
cybercat.....18, 25, 27, 28, 52, 53, 81, 83, 87
CYBERCAT.....1
Cybernetic.....3
CYBERSIM.....13, 71
cybersimulation.....19, 28, 54, 88
effect. .18, 19, 25, 27, 29, 31, 52, 53, 54, 55, 81, 83, 87, 92, 94
Effect.....19, 29, 32, 54, 56, 92, 95
Enter.....22, 49, 66, 113
hierarchy.....19, 31, 55, 94
Hierarchy.....19, 30, 55, 94
Interactions.....20, 36, 58, 99
interactive.....19, 33, 56, 96
Interpreter.....14
INTERPRETER.....71
language.....7, 20, 36, 58, 99
Menu.....20, 35, 57, 98

mice.....19, 30, 54, 93
model .18, 19, 25, 27, 28, 29, 52, 53, 54, 81, 83, 87, 92
Model.....3, 19, 29, 54, 92
Natural.....3, 19, 30, 55, 94
non-hierarchical.....19, 32, 56, 95
Paper.....3
path….18, 19, 27, 28, 30, 31, 53, 54, 55, 83, 87, 93, 94
Press.....22, 49, 66, 113
Print.....18, 27, 53, 87
prints.....18, 27, 53, 87
program….18, 19, 20, 22, 36, 38, 49, 52, 53, 55, 58, 59, 66, 67, 81, 83, 87, 94, 98, 99, 100, 113
PROGRAMMING.....71
proof.....20, 36, 58, 99
python.....7
PYTHON.....1, 71
query.....20, 36, 58, 99
shape.....19, 31, 55, 94
Show…18, 19, 20, 28, 29, 30, 32, 35, 54, 55, 56, 57, 88, 92, 94, 95, 98
Simulation.....3, 20, 38, 59, 100
SIMULATION.....71
state.....18, 25, 52, 81
statement.....20, 36, 58, 99
states.....18, 25, 52, 81
States.....18, 25
study……18, 25, 27, 28, 52, 53, 54, 81, 83, 87, 88
Summary.....19, 33, 56, 96
transition.....18, 25, 27, 28, 52, 53, 81, 83, 87
Transition.....18, 27, 53, 83
transitions.....19, 29, 54, 92
Transitions.....18, 27, 53, 87
transversing.....18, 27, 53, 87

Validate.....20, 36, 58, 99
verify.....18, 25, 27, 28, 52, 53, 81, 83, 87
Verify.....20, 36, 58, 99
views.....18, 28, 54, 88
Views.....18, 28, 54, 88

Typographical Conventions for None

Normal Text

```
1 1, cat, potential
2 2, potential, search
3 3, search, human
4 4, potential, chase
5 5, chase, mice
6 6, potential, symbol
7 7, symbol, operation
8 8, operation, intelligence
9 9, intelligence, physicale
```